Louisiana State University
LSU Digital Commons

LSU Doctoral Dissertations | Graduate School

6-8-2020

A Study on the Improvement of Data Collection in Data Centers and Its Analysis on Deep Learning-based Applications

Dipak Kumar Singh
Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

 Part of the Computer Engineering Commons

Recommended Citation

Singh, Dipak Kumar, "A Study on the Improvement of Data Collection in Data Centers and Its Analysis on Deep Learning-based Applications" (2020). *LSU Doctoral Dissertations*. 5285.
https://digitalcommons.lsu.edu/gradschool_dissertations/5285

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contactgradetd@lsu.edu.

A STUDY ON THE IMPROVEMENT OF DATA COLLECTION IN DATA CENTERS AND ITS ANALYSIS ON DEEP LEARNING-BASED APPLICATIONS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctorate of Philosophy

in

The Division of Computer Science and Engineering

by
Dipak Kumar Singh
B.S., National Institute of Technology, Durgapur, India, 2013
August 2020

Acknowledgments

I would like to acknowledge and thank my fellow students who have supported me and helped me with the work in this dissertation; Shayan Shams, Sayan Goswami, Richard Platania, Arghya Das and especially Chui-hui, with whom I worked on various projects and co-authored many papers. I would also like to thank my advisor, Seung-Jong Park, for supporting and motivating me to pursue and finish my Ph.D. I am also grateful to Seungwon Yang for mentoring me for my research work. Further thanks to other committee members, Jianhua Chen, Kisung Lee, and the Dean's representative Hunter Gilbert for guiding me through my research. Thanks to all co-authors who assisted in research with me.

I greatly appreciate the funding received from NIH LBRN grant (P20GM103424), NSF CC-NIE award (1341008), NSF MRI grant (MRI-1338051), NSF IBSS-L (1620451), NSF CISE grant CNS-1566443, Louisiana Board of Regents under grant LEQSF(2016-19)-RD-A-08, and LEQSF(2015-18)-RD-A-11, and National Science Foundation IBSS-L 1620451 to conduct this research.

Table of Contents

List of Tables

List of Figures

Abstract

Big data are usually stored in data center networks for processing and analysis through various cloud applications. Such applications are a collection of data-intensive jobs which often involve many parallel flows and are network bound in the distributed environment. The recent networking abstraction, coflow, for data parallel programming paradigm to express the communication requirements has opened new opportunities to network scheduling for such applications. Therefore, I propose coflow based network scheduling algorithm, *Coflourish*, to enhance the job completion time for such data-parallel applications, in the presence of the increased background traffic to mimic the cloud environment infrastructure. It outperforms Varys, the state-of-the-art coflow scheduling technique, by 75.5% under various workload conditions. However, such technique often requires customized operating systems, customized computing frameworks or external proprietary software-defined networking (SDN) switches. Consequently, in order to achieve the minimal application completion time, through coflow scheduling, coflow routing, and per-rate per-flow scheduling paradigm with minimum customization to the hosts and switches, I propose another scheduling technique, *MinCOF* which exploits the OpenFlow SDN. *MinCOF* provides faster deployability and no proprietary system requirements. It also decreases the average coflow completion time by 12.94% compared to the latest OpenFlow-based coflow scheduling and routing framework.

Although the challenges related to analysis and processing of big data can be handled effectively through addressing the network issues. Sometimes, there are also challenges to analyze data effectively due to the limited data size. To further analyze such collected data, I use various deep learning approaches. Specifically, I design a framework to collect Twitter data during natural disaster events and then deploy deep learning model to detect the fake news spreading during such crisis situations. The wide-spread of fake news during disaster events disrupts the rescue missions and recovery activities, costing human lives and delayed response. My deep learning model classifies such fake events with 91.47% accuracy and F1 score of 90.89 to help the emergency managers during crisis. Therefore, this study focuses on

providing network solutions to decrease the application completion time in the cloud environment, in addition to analyze the data collected using the deployed network framework to further use it to solve the real-world problems using the various deep learning approaches.

Chapter 1. Introduction

We live in the world of data. Due to the availability of the large amount of data, we need data centers for the analysis, storage and computation of such big data. The extreme large data can lead to lots of problem for performing certain tasks. For example, various cloud applications such as map-reduce, or deep learning applications, are carried out in phases, where each stage have many inter-communication flows, and only after the execution of the present stage, the application proceeds to the next stage. As a result, it can create delayed job completion issue due to the limited network resources such as bandwidth, or can cause job starvation due to the prioritization of larger flows in the network over the smaller flows. Traditionally, the network layers have little information about the application layer, however, with the introduction of coflow, the scheduling of jobs in data center networks have been improved. The coflow scheduling therefore plays an important role in finishing a job with minimum completion time. This report proposes such coflow scheduler to handle the minimum job completion time during the presence of the unknown background traffic to mimic an actual cloud environment. This way it decreases the job completion time of various cloud applications in data center networks, where the administrator has no information about the traffic generated by the client hosts. Furthermore, the report investigates the coflow scheduling, per-flow rate-limiting along with the coflow routing, to achieve better coflow completion time in storage area networks. However, the prior scheduling approaches requires customization of hosts and switches, and external commercial monitoring framework, which makes it difficult to be easily deployable in various cloud environment. To tackle this issue, the report presents a coflow scheduling/routing technique, *MinCOF*.

Finally, the report presents the data collected in data center networks for solving real world problems using deep learning applications. Specifically, it illustrates an end-to-end framework to collect twitter data in real time during natural disaster events and designs a model to classify such events as fake or real to tackle misinformation propagation during such crisis situations, which can help emergency responders to accomplish rescue missions suc-

cessfully and effectively.

1.1. Goal

The goal of this work is to deal with the data problem we face due to the availability of large amount of data in data center networks and/or the availability of small amount of data (such as during natural disaster events) in order to achieve the faster job completion time as well as effective analysis of the collected data to solve real world problems. We propose coflow scheduler to help alleviate the fast application completion time for big data applications to address the former issue, whereas we study deep learning with Transfer Learning technique to deal with the problem caused by small data to address the latter.

The technical strategy for the study is described as follows:

1) Develop coflow scheduling strategy to handle the big data job completion time problem in data center network environment, in the presence of uncontrolled clients hosts.
2) Develop coflow scheduling algorithm exploiting the OpenFlow SDN frameworks in storage area network environment to achieve cheaper and faster deployability and minimum customization.
3) Analyze the twitter data collected during natural disaster events in data center networks using the data collection system infrastructure.
4) Define the solution to the problem (use deep learning model with Transfer Learning to classify fake news during natural disasters).
5) Evaluate the performance of the solution.

1.2. Dissertation Outline

The remainder of this dissertation is as follows. Chapters 2 and 3 focuses on the strategy 1 and 2 respectively. Chapter 4 provide details for the strategy 3 and 4. Finally, Chapter 5 describes the conclusion and summary of the overall study and findings.

Chapter 2. Coflourish: A Coflow Scheduling Framework for Clouds

2.1. Introduction

Big data related researches have played a vital role in efficiently processing the data-intensive and parallel applications [17, 35]. Such applications typically consists of multiple inter-dependent stages. For instance, a MapReduce framework has Map, Shuffle and Reduce stages. The machines running such framework requires the data exchange in the previous stage to complete before processing the next stage. In data-intensive applications, such data exchange time can constitute to as much as 70% of the application completion time [10]. Therefore, shortening such data exchange time in each phase can greatly reduce the overall application completion time.

Normally, a data exchange consists of multiple parallel inter-communication flows between different machines in a cluster. Each stage of a data exchange finishes only when all the associated flows completes. Therefore, various strategies for reducing such data exchange time should consider all the associated flows as a logic unit. Coflow [8] is such network abstraction of the collective communication characteristics such as requirement and behavior between two groups of machines. Several coflow scheduling frameworks [9, 11] allows the developers to define the communication characteristics of applications using the coflow abstraction. Such characteristic definition helps in monitoring and manipulating the coflow transmission order and transmission rate, to shorten the coflow completion time (CCT). For instance, the existing coflow scheduling framework, Varys [11] makes the scheduling decision by using the feedback (the information of the available bandwidth of the network interface card (NIC)) from the daemon processes running on each controlled hosts. Varys is based on the assumption that the underlying network consists only of the non-blocking switches connecting all hosts.

However, such assumption is not feasible when the existing coflow scheduling frame-

© 2017 IEEE. Reprinted, with permission, C. Chiu, D. K. Singh, Q. Wang and S. Park, "Coflourish: An SDN-Assisted Coflow Scheduling Framework for Clouds," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 1-8, doi: 10.1109/CLOUD.2017.10.

works are implemented in a cloud environment. A cloud is typically a cluster of machines connected in a data center networks (DCN) [1, 23], shared among multiple tenants such as Amazon EC2 [4] and Google Cloud Platform [21]. Each tenants are allowed to deploy their own customized software environment on assigned hosts or machines. As a result, the existing cloud scheduling framework cannot guarantee that their daemons are running on those hosts to send the network feedback for performing scheduling tasks. Such uncontrolled hosts can run various applications which may generate complicated communication flows in the cloud network. We refer to such category of uncontrolled communication flows as the background traffic in the rest of this paper. The existing coflow network are unaware of the network bandwidth consumed by such background traffic. As a result, when the background traffic (such as high-definition video streaming [5]) grows in the network rapidly, such framework starts to perform poorly.

In this chapter, we study the performance degradation of such existing coflow scheduling framework such as Varys in the cloud environment, and propose a solution to improve the performance during the presence of background traffic by exploiting the network feedback from the switches in the DCN.

The motivation of the study is evaluation of the performance loss resulted due to the background traffic. We implement a trace-drive simulation of the Varys coflow scheduling framework [11]. The simulation considers network workload generated using the Facebook traffic probability distribution . We observe up to 82.1% decrease of CCT (section 2.3). We conclude that the in-network bandwidth feedback information is crucial to schedule the coflows effectively in such environment.

We propose *Coflourish*, a coflow scheduling framework which aggregates the congestion feedback from the SDN-enabled switches in the fat-tree based DCN [1, 23] for correct estimation of the available bandwidth in the network. It is achieved by sending the overall congestion information of a switch (by tracking the congestion at each port) at each layer to each host machine (section 2.4). A daemon process running on each host collects the feedback information

and maps it into available bandwidth from(to) itself to(from) each fat-tree layer and report the available bandwidths to a logical centralized scheduler. The scheduler then analyzes the collected bandwidths information and uses the minimum of the available bandwidth between a source and destination as its overall available bandwidth to schedule the coflows. *Coflourish* is the first coflow scheduling framework which provides a detailed algorithms of the SDN-assisted available bandwidth estimation.

To evaluate *Coflourish*, we create a trace-drive simulation and run it on various DCN workloads in the flow level. In the presence of background traffic, the simulation results show that *Coflourish* estimates the available bandwidth 4.1% more accurately than Varys under the smaller background traffic, and 78.7% more accurately under the larger background traffic. The results also reveal that our framework reduces the average CCT up to 75.5% compared to Varys (section 2.5).

2.2. Related Work

To enhance the average application completion time, three broader scheduling frameworks has been studied; **coflow-based, flow-based and load balance based**.

Varys [11] is the most representative coflow-based scheduling framework. It uses the SEBF heuristic algorithm to reduce the average CCT. It works under the assumption that the underlying switches consists of non-blocking switch connecting to all hosts. Each host runs a daemon to find the available network bandwidth and schedule the coflow based on such information in between the hosts. Rapier [58] expands the functionality of Varys by also taking the coflow routing information into consideration. A framework which feeds routing and network utilization information to the scheduler is assumed to exist. However, both of such framework suffers during the presence of the background traffic in the cloud environment.

Baraat [18] is a flow-based scheduling framework which focuses on scheduling the flows in the network to achieve the minimum flow completion time (FCT). It gives higher priority to flows with smaller cumulative size to prevent smaller flows to starve in the presence of larger flows. It injects task ID information into the network traffic. The flows with same priority,

are served on First-in-First-out (FIFO) basis according to the value of the task ID. pFabric [3] prioritizes flows based on its size at each switch. However, flow-based frameworks are unaware of the logical relationship between the flows in the application level. Such flow-based scheduling without the application level information may lead to a sub-optimal completion time.

The load balancing strategy balances the load of the traffic by equally distributing it across the network. CONGA [2] allocates the traffic to the least congested link based on the congestion status at each port (using DRE), it stores between each leaf-to-leaf switch in a fat-tree based DCN. However, such balancing technique also relies on the flow information for its load balacing and hence, carries the same disadvantages as the flow-based scheduling framework.

Coflourish is a coflow-based scheduling framework, which overcomes the disadvantages of the Varys and Rapier. It further utilizes the DRE for its congestion estimation at each switch port.

2.3. Motivation

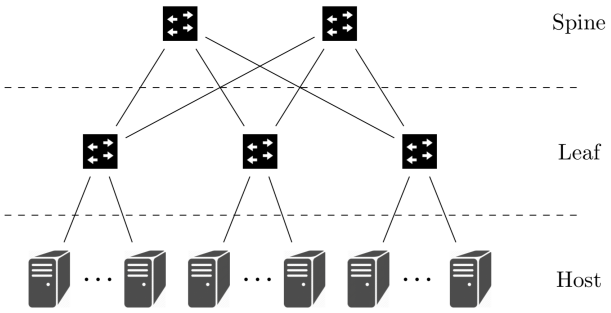


Figure 2.1. A spine-leaf fat-tree DCN with 2 spine switches and 3 leaf switches. Each switch in the leaf layer connects to the same number of host machines.

To study how the existing coflow scheduling framework is impacted by the network congestion in the presence of background traffic in clouds, we use Facebook trace driven simulation. We analyze it using the spine-leaf fat-tree [2] as shown in Figure 2.1. The details of the simulation configurations are given in section 2.5.1.

The simulation traffic contains two types of traffic loads, regular size traffic load (Regular) and large size traffic load (Large). The Regular loads consists of flows involved in an usual

cloud applications such as the Map Reduce jobs, where as the Large loads consists of 50% of the load generated using the data intensive jobs such as HD video streaming or the deep-learning applications and the remaining 50% comprises of Regular loads. In order to inject the background traffic in the network, we remove the coflow related information from the trace flows. The weight of the background traffic is then varied from 10% to 90% with 10% interval. We compare the performance of Varys with respect to the ideal scheduling framework similar to Varys, which has precise network bandwidth information, and whenever the network congestion in a link changes, the coflow is scheduled based on the updated congestion information.

By comparing the Varys with the ideal scheduling framework (using the average CCTs normalized to Varys using Eq. 2.1) as shown in Figure 2.2, we see that as the weight of the background traffic increases, the averages CCT performs poorly. The ideal scheduling framework achieves 4.1%(82.1%) decrease in average CCT with 10%(90%) background traffic under the regular size coflow traffic load and achieves 7.3%(80.2%) decrease in average CCT with 10%(90%) background traffic under the large size coflow traffic load.

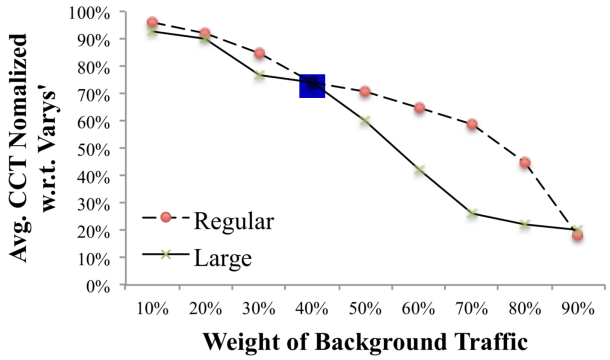


Figure 2.2. CCT of the ideal scheduling framework normalized to Varys.

$$\text{Normalized CCT} = \frac{\text{New Fwk's CCT}}{\text{Vary's CCT}} \tag{2.1}$$

2.4. Framework Design

We propose *Coflourish*, which improves upon the average CCT when compared to the existing coflow scheduling framework such as Varys, in the presence of the background traffic. The design goals of *Coflourish* are as follows:

Switch-assisted Bandwidth Estimation: The status of the available network traffic information should be monitored by the switches in the DCN without imposing complicated computations on them and are reported to the centralized coflow scheduler.

Starvation Prevention: Coflow should not wait for a long time at the queue because of the lack of network resources.

Work Conservation: Coflow should be allocated any necessary resources it needs for further computation.

The overall *Coflourish* scheduling framework is described in the subsequent sections in detail:

2.4.1. Framework Workflow

Coflourish exploits the information exchange between various components of a DCN such as network switches, host machines, and a logically centralized coflow scheduler to achieve the design goals.

1) Each spine switch in the DCN sends their congestion information periodically to each leaf switch which it connects to (section 2.4.2).

2) Each leaf switch then accumulates all the congestion information it receives from the spine switches along with it's own per-port congestion information and sends it to the host it connects to (section 2.4.2).

3) Each host then calculates the available network bandwidth information based on the accumulated information from the leaf and spine switches, or estimates any missing information, and sends the bandwidth information to the centralized scheduler (section 2.4.3).

4) The coflow scheduler then uses the bandwidth information of the network, which also captures the network information caused by the background traffic (as reported by the network

Table 2.1. Symbols used in algorithms.

Symbol	Description
CT	Congest table on current switch
CT_p	Congestion value in the CT for Port p
$Pt(X,Y)$	Port on Switch X connecting to Switch Y
$AB(X)$	Available bandwidth of Port X
P_i^{out}	egress port on Machine $i(H_i)$
P_i^{in}	ingress port on H_i
S	Set of all switches in the spine layer
s	Total elements in S
L	Set of all switches in the leaf layer
l	Total elements in L
H	Set of all host machines in the host layer
h	Total machines a leaf switch connects to
$X_{y..z}$	Element with Index y through z in set X
UL_i	Avl. BW to leaf layer switch to H_i
DL_i	Avl. BW from leaf layer switch to H_i
US_i	Avl. BW to spine layer switch to H_i
DS_i	Avl. BW from spine layer switch to H_i
C	Network link capacity

switches) to estimate the available bandwidth between each source and destination machine pair and schedules the coflows (section 2.4.4).

2.4.2. SDN-enabled switch

Coflourish implements its scheduling framework using the custom configurations on the network links as well as the data structures and algorithm for the switches. The chapter uses many symbols as represented in the Table 2.1. The DCN topology used for the algorithm is shown in the Figure 2.3. For the sake of simplicity, only one switch L_n is depicted in the figure, connecting to the host machines, however, in reality, all the leaf switches are connected to the host machines in a fat-tree topology. We consider each network link have identical capacities.

Each link in the topology is further divided into two channels. One channel carries the network congestion information to avoid any delay in the transmission of such information for the scheduling purposes, and the other channel is used for the actual data packets transfer. This bifurcation can easily be deployed using the Priority Code Point (PCP) in the Ethernet.

Furthermore, each switch maintains a congestion table and algorithms for updating the table and the updated information is then transferred from the leaf layer to the host layer.

Each entry of the congestion table holds the congestion information calculated at each switch port and is updated using the ESTCONG (in Algorithm 1). Whenever a packet passes through a port, the congestion value of that port is incremented by the size of the packet (Line 2-4) or decremented by a factor of $(1-\alpha)$, where $\alpha \in (0,1)$, every T_{dre} interval (Line 5-9). This algorithm is similar to the Discontinuing Rate Estimator (DRE) used in [2], without the computation of the congestion metric. However, such computation can be deployed along with *Coflourish* to provide coflow scheduling and load balancing mechanism.

Algorithm 2 shows how the congestion information flows from one layer to another every

Algorithm 1 Congestion Estimation

```
1:  procedure ESTCONG(Event E, CT)
2:      if E is packet arrival then
3:          p ← event source switch port
4:          CT_p ← CT_p + d
5:      else if E is decrease timeout then
6:          for all CT_x ∈ CT do
7:              CT_x ← (1 − α)CT_x
8:          end for
9:          Schedule a decrease timeout in T_dre interval
10:     end if
11: end procedure
```

T_{FB} interval. Each switch in a layer, accumulates all the congestion information from the top layer switches and sends this information along with it's per-port congestion value to the bottom layer. For instance, in case of the topology, shown in Figure 2.1 the switches in the spine layer runs the FBSPINE (in Algorithm 2), where they sends the congestion information from spine-to-leaf to each leaf layer switch (L_x) it connects to (Line 3-4). Consequently, each leaf layer switch then runs the AGGFBLEAF (in Algorithm 2) to sends three pieces of information from leaf-to-host to the host machine H_x it is connected to. First information carries the summation of all the from-spine congestion values from all spine switches (Line 9). Second information holds the accumulated congestion information at each leaf layer from all the spine

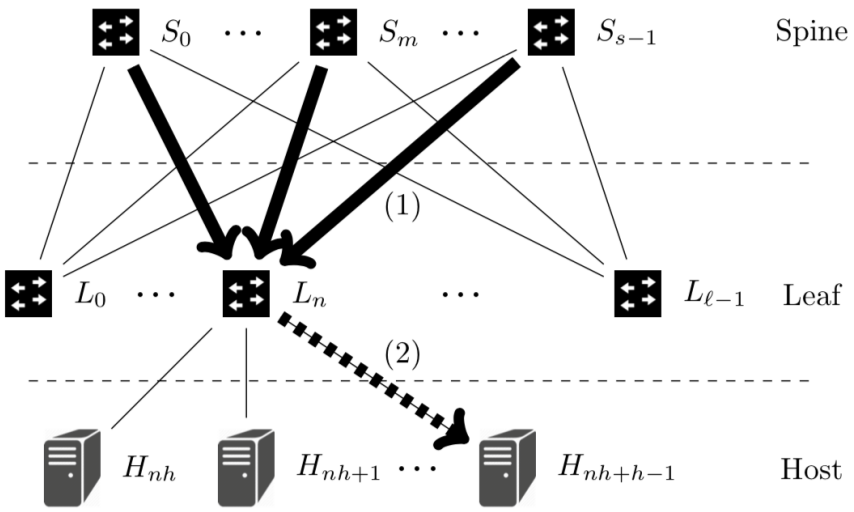


Figure 2.3. Switch feedback mechanism to machine H_{nh+h-1}. Phase (1): spine layer feedback to leaf layer (solid arrow). Phase (2): leaf layer feedback to host layer (dashed arrow)

switches in the top layer (Line 10-12), and third information contains the congestion value of the each egress port at the leaf switch which connects it to host machine H_x (line 15-17). We only consider the congestion at egress ports. It is because the ingress port must be the egress traffic which is shaped by the congestion at some egress port, therefore, the congestion happens only at the egress ports. By designing such framework, helps us to achieve the minimal customization of the switches leading to little overhead at the switch layer, and major complexities are handled on the host machines, and the coflow scheduler.

To reduce the traffic load introduced by the feedback mechanism, each switch send feedback messages using simple transmission protocols such as Point-to-Point Protocol (PPP) and High-level Data Link Control (HDLC).

To implement such mechanism on the switches in real cloud environment, SDN-enabled switch can be used. Such SDN-enabled switch can either be physical switch or logical switch on the host machine to connect multiple virtual machines. For example, a physical switch which support the P4 programming language [15] or the Open vSwitch software switch can be easily deployed in production cloud management systems [13].

2.4.3. Host Machine

The host machine in *Coflourish* is responsible for calculating the smallest available bandwidth from(to) itself to(from) each switch layer, which can later be used by the coflow scheduler for estimating the available bandwidth between a source-destination pair. It is also responsible to transfer the flows at the transmission rate feedback directed by the scheduler to the host after the coflow scheduling takes place.

In order to calculate the smallest available bandwidth, the host runs AGGRPTHOST (in Algorithm 3). The host calculates the available bandwidth using the Eq. 2.2, where it utilizes the fact that the congestion value (Cng) is proportional to the bandwidth of a traffic [2].

$$Cng = \lambda(C - A)\tau \implies \lambda\tau = \frac{Cng}{C - A} \implies A = C = \frac{Cng}{\beta} \tag{2.2}$$

where C is the capacity of the link, λ is a scaling factor, τ is a time constant, and $\beta = \lambda\tau$ is a scaling factor. The from-leaf-to-host congestion value is fed back from leaf layer to the host machine (fS in Line 5). The host machine also knows its own exact available bandwidth from the leaf switch to itself (Line 4). Thus, the β in Eq. 2.2 is calculated (Line 6). The leaf-to-spine available bandwidth is calculated (Line 7). The tS is an aggregated congestion value of all network links from a leaf switch to all s spine switches so that the link capacity used in Eq. 2.2 should also be aggregated as sC. The bottleneck available bandwidth from current machine to spine layer is calculated (Line 8). The bottleneck available bandwidth from spine layer to current machine can be calculated through a similar process (Line 9-10). Finally, The available bottleneck bandwidths from current machine to each switch layer are reported to the *Coflourish* scheduler using a reliable transmission protocol such as TCP.

In case of some congestion information misses along their paths to the machine because of the simple transmission protocol in use, we deploy a Kalman Filter [30] for each congestion value in the feedback mechanism to estimate the latest values. We simplify the congestion value estimation into a motion model with constant acceleration and measurement noise. All

state variables of the filters are updated when its corresponding congestion value is successfully fed back to the host machine.

For controlling the rate of transmission of flows at each host, the daemon process of *Coflourish* throttles each flow using the mechanism similar to Varys based on the rate it receives by the coflow scheduler.

Algorithm 2 Congestion Feedback & Aggregation

```
1: procedure FBSPINE(CT, L)
2:     for all L_x ∈ L do
3:         pe ← Pt(current switch, L_x)
4:         fS ← CT_pe                                  ▷ S to L cong.
5:         Send fS to L_x
6:     end for
7: end procedure

8: procedure AGGFBLEAF(Feedbacks FB, CT, S, H)
9:     fS ← ∑(fS ∈ FB)                                 ▷ Agg. S to L cong.
10:    for all S_x ∈ S do
11:        pe ← Pt(current switch, S_x)
12:        tS ← ∑ CT_pe                                ▷ Agg. L to S cong.
13:    end for
14:    n ← current leaf switch ID
15:    for all H_x ∈ H_{nh..nh+h−1} do
16:        pe ← P(current switch, H_x)
17:        fL ← CT_pe                                  ▷ Agg. L to H cong.
18:        Send fS, tS, fL to H_x
19:    end for
20: end procedure
```

2.4.4. Coflow Scheduler

The coflow scheduler of *Coflourish* decides the order in which the coflows are scheduled, as well as the transmission rate of the flows of each coflow at each host. We enhance the start-of-the-art existing algorithm in Varys to further implement our scheduler. This section describes the details of the scheduler, to help achieve the design goals.

To provide switch-assisted bandwidth estimation, a PathRem array is maintained at the scheduler as shown in Table 2.2. The table tracks the bi-directional available bottle bandwidth

Algorithm 3 Congestion Aggregation & Report

procedure AGGRPTHOST(Feedback FB)
  $i \leftarrow$ current machine ID
  $UL_i \leftarrow AB(P_i^{out})$ ▷ BW to Leaf
  $DL_i \leftarrow AB(P_i^{in})$ ▷ BW from Leaf
  $fS, tS, fL \leftarrow fS, tS, fL \in FB$
  $\beta \leftarrow fL/(C - DL_i)$
  $LtoSBW \leftarrow sC - \frac{tS}{\beta}$
  $US_i \leftarrow \min(LtoSBW, UL_i)$ ▷ BW to Spine
  $StoLBW \leftarrow sC - \frac{fS}{\beta}$
  $DS_i \leftarrow \min(StoLBW, DL_i)$ ▷ BW from Spine
  Send US_i, DS_i, UL_i, DL_i to scheduler
end procedure

from(to) each host machine to(from) each switch layer. PathRem(i, j, h) represents the smallest bottleneck bandwidth from host i to host j, where h is equal to the total number of machines to a single leaf switch at each rack. PathRem first, finds all the possible path between host and destination machines through the switch layer using the Algorithm 3. It then calculates the upward and downward available bandwidth through all such feasible paths, and finally deduces the optimum available bandwidth between the source and destination hosts using the Eq. 2.3.

$$PathRem(i, j, h) = \begin{cases} \min(UL_i, DL_j) & \text{if (i } \textbf{div} \text{ h) = (j } \textbf{div} \text{ h)} \\ \min(US_i, DS_j) & \text{otherwise} \end{cases} \quad \text{where } \textbf{div} \text{ is the integer division.} \tag{2.3}$$

Algorithm 4 shows the extension of the Varys, and how coflows are scheduled using the SERVECOFLOW procedure. Only coflows with size greater than 25MB are scheduled. SERVECOFLOW comprises of two parts. First part achieves the average CCT minimization and work conservation, and the second part achieves the starvation prevention of the design goals. Each time a new coflow arrives/finishes or the available bandwidth exceeds a certain threshold, SERVECOFLOW is restarted and generates a new PathRem table entry to schedule the coflows.

Varys requires each host to run a daemon which estimates the effective bottleneck for each coflow by utilizing the estimated available end-to-end bandwidth of the NICs across

Table 2.2. The PathRem which stores bi-directional available bottleneck bandwidths at the scheduler.

Direction	Layer	Host			
		H_1	H_2	...	H_{lh-1}
Up	Leaf	UL_1	UL_2	...	UL_{lh-1}
	Spine	US_1	US_2	...	US_{lh-1}
Down	Leaf	DL_1	DL_2	...	DL_{lh-1}
	Spine	DS_1	DS_2	...	DS_{lh-1}

DCN. Therefore, the coflows with the Smallest Effective Bottleneck First (SEBF) heuristics are scheduled in that order (Line 3). However, in cloud environment, the administrators do not have control over several hosts, as a result, Varys fails to estimate the background network traffic generated by such uncontrolled host, and schedules the coflows based on partial information of the network and performs poorly.

Coflourish further enhances the Varys algorithm, by using the Eq. 2.3 **to minimize the average CCT**. In Case 1, if both the source machine i and the destination machine j are connected to the same leaf switch, then the available bottleneck bandwidth is the minimum of the bandwidth between the leaf switch to the destination host (DL_j) and between the source machine to leaf switch (UL_i). However, if two end hosts are connected to different leaf switches via a spine layer (Case 2), then the end-to-end available bandwidth is the minimum of the available bandwidth to the spine layer from machine $i(US_i)$ and the available bandwidth from the spine layer to the machine $j(DS_j)$. As a result, we calculate the average CCT in both directions (upward and downward link) using the Eq. 2.4 and 2.5 respectively. Consequently, the longest CCT (Γ) is determined as the bottleneck of a coflow by taking the accurate end-to-end available bandwidth as shown in Eq. 2.6. Thus, the coflows are scheduled using the shorted bottleneck heuristic (SBF).

$$CCT_f = \max_i \left(\max_j \frac{d_{ij}}{PathRem(i,j,h)} \right) \tag{2.4}$$

$$CCT_b = \max_j \left(\max_i \frac{d_{ji}}{PathRem(j,i,h)} \right) \tag{2.5}$$

where $d_{ij}(d_{ji})$ is the summation of the remaining data size of the current coflow from Machine $i(j)$ to Machine $j(i)$.

$$\Gamma = \max(CCT_f, CCT_b) \tag{2.6}$$

To provide work conservation, we allocate the remaining bandwidth (if any) to the coflows which are ready to be transferred in the minimum schedule (Line 14).

To prevent starvation, all coflows which were not allocated in the previous stage, are allocated the bandwidth. This is achieved by alternating between the CCT minimization stage and starvation prevention stage for every T_{sel} and T_{str} time interval respectively. This strategy guarantees that each coflow progresses for at least T_{str} interval every $T_{sel} + T_{str}$ interval and therefore, is not starved (Line 24-31).

To adapt to the sudden change in the network due to the variation in the background traffic, we re-invoke the SERVECOFLOW procedure if it exceeds a tunable threshold parameter, $thld_{BW}$ (Line 32-35). This captures the significant change in the network behavior and generates a new schedule. $thld_{BW}$ determines the sensitivity of *Coflourish* to available bandwidth change in the DCN.

2.5. Evaluation

We extend the trace-driven flow-level simulation created in section 2.3 to evaluate the performance of *Coflourish* .

2.5.1. Simulation Framework

Our simulation traffic load consists of two kinds of coflows, regular and large coflows. Regular coflows are generated using the Facebook traffic probability distribution used in Varys [11]. Large coflows is defined as a coflow size uniformly distributed between 25MB and 1GB. In *Coflourish*, only large coflows are considered for scheduling task. This is to reduce the scheduling overhead. Once the traffic load is generated, various percentage of large coflows are injected into the network. Specifically, we consider three different percentages of large coflows: 25%, 50% and 75% in our simulation. In addition to this, we also add different per-

Algorithm 4 Scheduling & DCN Bandwidth Monitoring

```
1: procedure SHAREBW(Coflows γ, PathRem(.))
2:     for all coflow C in γ do
3:         Calcuate Γ using Eq 2.6
4:         for all flow in C do
5:             rate ← (flow's remaining size)/Γ
6:             Update PathRem(i,j,h) with rate
7:             Update PathRem(j,i,h) with rate
8:         end for
9:     end for
10: end procedure

11: function SELECT(Coflows γ, PathRem(.))
12:     C_sel = Sort all Coflows in γ in SBF order
13:     ShareBW(C_sel,PathRem(.))
14:     Assign remaining BW to coflows in C_sel
15:     C_str = starved coflows in C_sel
16:     return C_sel, C_str
17: end function

18: procedure SERVECOFLOW(Coflows γ, PathRem(.))
19:     if is CCT min. stage then
20:                                                    ▷ CCT min. stage
21:         Stop coflows in C_str
22:         C_sel, C_str = SELECT(γ, PathRem(.))
23:         Switch to feeding starved stage in T_sel interval
24:     else if is starvation previous stage then
25:                                                    ▷ starvation previous stage
26:         Stop coflows in C_sel
27:         for all coflows in C_str do
28:             Add all flows to one_coflow
29:             ShareBW(one_coflow, PathRem(.))
30:             Switch to min CCT stage in T_str interval
31:         end for
32:     else if net available BW change > thld_BW then
33:                                                    ▷ Significant available BW change
34:         C_sel, C_str = SELECT(γ, PathRem(.))
35:         go back to interrupted point of execution
36:     end if
37: end procedure
```

Table 2.3. Default framework parameter configuration.

Parameter	Value
T_{FB}	200 milliseconds
T_{sel}	2 seconds
T_{str}	200 milliseconds
$thld_{BW}$	1/8 spine layer network link capacity
ρ	500 microseconds
α	0.5
T_{dre}	250 microseconds

centages of background traffic in the network. Such background traffic are generated by removing the coflow attributes from the traffic load. The weight of the background traffic varies from 10% through 90% with a 10% interval. We generate 3000 host machines connected by a spine-leaf fat-tree DCN. Each leaf switch connects to 50 host machines. The oversubscription is 10:1. The total number of coflows in our traffic load is 1000. The coflow arrival is a Poisson process with the Arrival Intensity 5. Other default framework related parameter configuration is presented in Table 2.3. Furthermore, the evaluations of the *Coflourish* on various weights of background traffic is empirically studied by taking the average of the results over 5 runs, and a mathematical proof can be further investigated to support the results.

2.5.2. Improvement of Coflourish

We estimate the efficiency of *Coflourish* based on the accuracy of the available bandwidth estimation and the improvement of the average coflow completion time.

To estimate the available bandwidth estimation accuracies, we use regular coflows and vary the weight of the background traffic. The metric, Normalized Inaccuracy, is defined in Eq. 2.7.

$$\text{Normalized Inaccuracy} = \frac{|\text{New Fwk's Estimated Error}|}{|\text{Varys' Estimated Error}|} \tag{2.7}$$

where the Err. is the difference between the estimated available bandwidth by a framework and the true available bandwidth in the network. The result (lower Normalized Inaccuracy is better) is shown in Figure 2.4(a). The *Coflourish* estimation is 4.1% (10% background) through

78.7% (90% background) more accurate than the Varys. When the weight of the background traffic is small, then the partial network bandwidth estimation using the simple NIC-based feedback in Varys is similar to the actual available bandwidth. As a result, Varys does not suffer from a significant performance loss. However, when the weight of background traffic increases due to the presence of uncontrolled hosts, Varys performance decreases significantly due to it's over dependence on the NIC-based network bandwidth estimation. However, *Coflourish* scheduling framework is based on the congestion feedback from the network switches. Hence, an accurate estimation of the network traffic is estimated.

Secondly, to estimate the improvement of the average CCT of *Coflourish* over Varys, we use regular traffic load and vary the weight of background traffic. The result is shown in Figure 2.4(b). The *Coflourish* shortens the average CCT by 4.3% (10% background) through 75.5% (90% background) compared to Varys. It can be noted that Varys performs well during the presence of smaller background traffic, however the average CCT in Varys increases significantly when the weight of the background traffic increases. This is because of the same reasons as described in the previous paragraph. Varys estimates the bandwidth based on the NIC-based report of the controlled hosts. On the other hand, *Coflourish* performs remains stable with the increase of the background traffic as it is accumulating the bandwidth estimate from the network itself than relying on the host daemons.

2.5.3. Impact of Coflow size

As described in the simulation configuration section, we consider 25%, 50%, and 75% large coflows respectively. The weight of background traffic varies from 10% through 90% with a 10% interval. The result is shown in Figure 2.4(a).

When the weight of background traffic is less than 30%, Varys still generates comparable schedule to *Coflourish* .

However, with the increase of the background traffic from 30% through 80%, background traffic heavily interferes with the bandwidth estimation of Varys. Average CCT increases faster. In contrast, *Coflourish* still generates good schedule with available bandwidth feedback infor-

mation. The more the large background traffic, the larger the average CCTs differ between the Varys' schedule and *Coflourish* 's schedule.

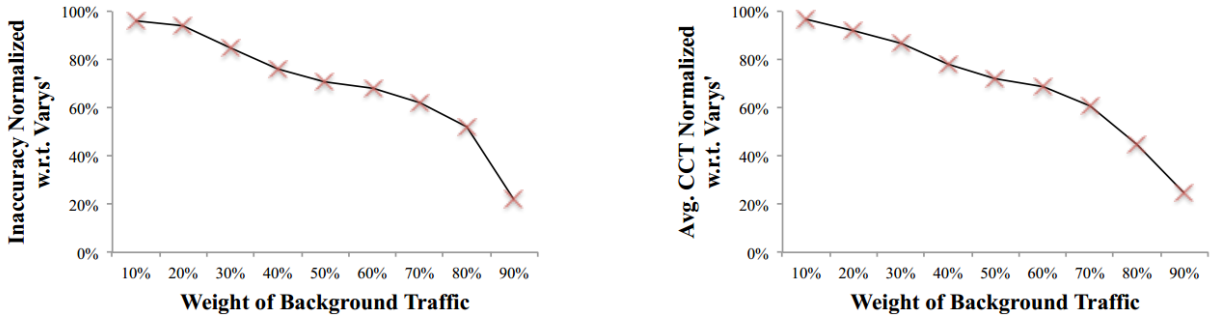
It is interesting to note that when the weight of the background traffic exceeds 80%, the gap between the average CCT in *Coflourish* with different large coflow load decreases. This is because of the saturation of the network, where the lower coflow transmission rate is not impacting the average CCT significantly.

2.5.4. Impact of Feedback and Bandwidth Variation Threshold

The impact of two critical parameters in *Coflourish*, the feedback interval (T_{FB}) and the bandwidth variation threshold ($thld_{BW}$) is investigated in this section.

The T_{FB} is varied to 100, 200, and 400ms, along with the variation in the weight of background traffic. The result is shown in Figure 2.5(b). A smaller A smaller T_{FB} means frequent update of the available bandwidth information to the scheduler. Therefore, the up-to-date PathRem table, mimics the optimal network bandwidth estimation, resulting in better average CCT, and vice versa.

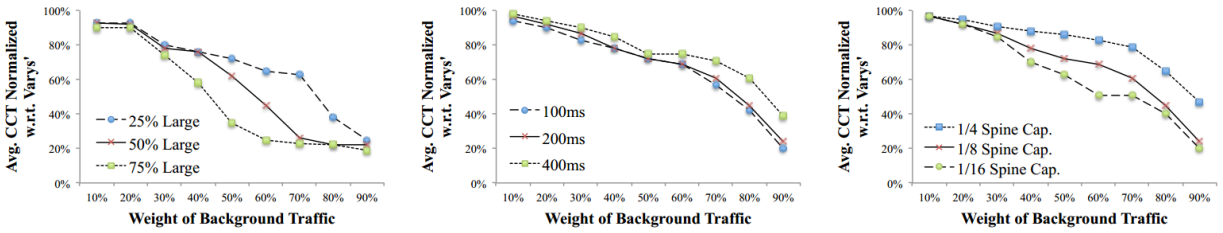
The ($thld_{BW}$) is set to 1/4, 1/8, and 1/16 the spine layer network link capacity, along with the variation of the weight of background traffic. The result is shown in Figure 2.5(c). A smaller ($thld_{BW}$) results in more frequent invocation of the coflow scheduling algorithm (SERVECOFLOW), as a result, an up-to-date available bandwidth is available which is closer to the optimal solution. A large $thld_{BW}$ results in the opposite.



(a) Inaccuracy

(b) CCT

Figure 2.4. Improvement of Coflourish normalized to Varys.



(a) Impact of coflow size (b) Impact of BW Feedback Interval (c) Impact of BW variation threshold

Figure 2.5. Coflourish with various configurations.

2.6. Conclusion

The existing coflow scheduling framework performs poorly in the presence of increased background traffic, as it is oblivious to the amount of network generated by the uncontrolled hosts in the shared cloud environment. To overcome such limitations, *Coflourish*, a coflow scheduling framework is presented. It exploits the network congestion information from each switch in the fat-tree topology, rather than relying on the NIC-based bandwidth calculation at each hosts, to schedule the coflows based on the accuration available bandwidth estimation. Simulation results reveal that *Coflourish* is 78.7% more accurate in terms of bandwidth estimation and 75.5% better in terms of average CCT than the existing framework. It can also be easily deployed along with the load-balancing capabilities in the underlying network during to its DRE-based integration.

Chapter 3. MinCOF: A Coflow Routing and Scheduling Framework for Storage Area Networks

3.1. Introduction

Data centers provides a feasible solution to process large amount of data, using the big data applications which runs on multiple computing resources available in the cluster. Such applications (such as Map-Reduce jobs) are usually divided into various phases. At each phase, a large number of communication flows are transferred between hosts in the cluster to finish a phase of the application. Only after a phase has completed, the application progresses to the next phase [17, 35]. Therefore, these frameworks transform applications into a mutual blocking computing and communication phases until all phases are successfully executed.

Usually, a cluster application invests a large amount of time at each communication phase. For instance, the intermediate data sizes of deep learning applications , or the Map-Reduce jobs of a communication phase are large. Therefore, in order to achieve faster application completion time, it is important to reduce the communication time of each phase. [8, 11] has introduced a coflow scheduling framework to achieve this goal. Coflow is a logical semantic for a collection of flows which shares a common set of attribute values. Therefore, by scheduling the coflows of an application at each phase, the overall application completion time is reduced [10]. Furthermore, different strategies to also incorporate the coflow routing information along with coflow scheduling has been found to achieve even better application completion time [58].

There are three important features to achieve the goal of a faster application completion time:

1) **Coflow Scheduling:** Determining the order of coflows and the rate at which each flows in a coflow are transmitted from all hosts.

© 2017 IEEE. Reprinted, with permission, C. Chiu, D. K. Singh, Q. Wang, K. Lee and S. Park, "Minimal Coflow Routing and Scheduling in OpenFlow-Based Cloud Storage Area Networks," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 222-229, doi: 10.1109/CLOUD.2017.36.

2) **Coflow Routing:** Determining the path, a flow should take to reach to it's destination hosts.

3) **Per-flow Rate-limiting:** Assigning a rate to each flow by a coflow scheduler to utilize the bandwidth between multiple coflows efficiently.

Some existing framework such as Varys [11] provides the coflow scheduling and/ rate limiting features. It achieves it by customizing the operating system (OS) on hosts to synchronize the coflow information to the coflow scheduler. However, if there are varieties of OS in a cloud, then running such network monitoring daemon at each host may be difficult to deploy or not feasible. Other existing framework such as Tailor [29] provides all the three features, by using a proprietary commercial monitoring tools (sFlow-RT [16]) on switches to synchronize the coflow information. However, deploying such proprietary tools on production environment incurs extra cost, and software-related expertise.

To achieve easy and cheaper deployability and minimal customization, *MinCOF*, a coflow based routing and scheduling framework is presented in this chapter. It exploits the existing standardised mature commodity OpenFlow Switches to provide all the three features. *MinCOF*incurs lesser overhead on the network switches/hosts by assigning most of the network monitoring task to the OpenFlow 1.3 Switch and a central coflow scheduler (section 3.4). It is especially suitable for cloud storage area networks (SANs) for storing big data. For example, the scientific cloud object storage [20, 24] involves massive coflow communication scenarios (section 3.3), and its big data transmissions which utilize few concurrent long flows can be processed by current OpenFlow SDN implementations. *MinCOF* is also easily adaptable to the existing SANs, with the addition of few lines of code to their original connection establishment and progress reporting functions.

MinCOF is evaluated on physical OpenFlow Switches and hosts on a data center networks (DCN) testbed. It largely outperforms Varys, and achieves 12.94% better coflow completion time (CCT) as compared to Tailor, without rate limiting, under various workload traffic and network topologies.

Table 3.1. Comparison of features of coflow scheduling/routing frameworks.

Framework	Coflow Scheduling	Coflow Routing	Per-flow rate-limiting	Unmodified Host OS	Coflow Info. Sync.
Varys	✓		✓		2-way
Rapier	✓	✓	✓		2-way
Tailor	Limited	1 flow		✓	Proprietary
MinCOF	✓	1 coflow	✓	✓	1-way

3.2. Related Work

Varys [11] is a coflow scheduling and rate limiting framework which requires customization of OS to run daemon to monitor the network interface usage. It then estimates the bi-directional (2-way) available network bandwidth from hosts to a network layer and sends it to the coflow scheduler for making the scheduling decisions.

Rapier [58] inherits all the features such as monitoring network interface usage, synchronizing coflow information and enforcing rate limiting decisions. In addition, it also implements coflow routing. However, it also requires customization of OS.

Tailor [29] is another coflow scheduling/routing framework built upon OpenFlow-based DCN. It adopts sFlow-RT, propreitary network switch monitoring tool to collect the coflow information. Per-flow rate-limiting is not considered. It also do not need any customization of OS. Therefore it is easily deployable.

MinCOF utilizes all the features of the existing framework, and gets rid of undesirable requirements such as customized OS, and propreitary system component. An OpenFlow-based DCN and extended transfer applications for the one-way coflow information synchronization are the only requirements.

Table 3.1 shows a comparison between all the exisiting framework and the features each one supports.

3.3. Motivation

In the previous chapter, it is showed how coflow scheduling framework can greatly enhance the application completion time in a cloud environment. Consequently, SAN is an-

other cloud like environment, which have similar fat-tree based topologies [1] and underlying communication patterns [8]. Like DCN, SANs also involves a group of hosts exchanging data or communication flows between other group of hosts. Therefore, a file transmission in SAN completes faster when all parallel communication channels completes at relatively minimal time, as the slowest flow determines the overall file completion time. Therefore, it is interesting to study how coflow routing/scheduling techniques can help data exchange/replication in SANs.

Another important motivation is the easy adaptability of OpenFlow SDN in the Cloud SAN for processing the traffic load. An OpenFlow SDN flow table can easily handle a large number of concurrent flows. Moreover, there are fewer flow arrivals and completions in SANs. When SAN stores big data, the total files are relatively few and individual files are typically large so that the parallel flows carrying those files are long-lived and can finish in comparable time. The undesirable overheads of processing those events in the OpenFlow SDN are less encountered.

Thus, the improvement brought by the coflow scheduling/routing technologies can help achieve better job transfer time in SANs as well. Also, because cloud SAN typically serves limited number of concurrent flows and experiences less frequent flow arrival/completion time, as compared to DCN for cluster computing, OpenFlow SDN for processing traffic load can be very effective.

Hence, the above observation suggest that incorporating the coflow scheduling and routing to OpenFlow-based cloud SANs, can help file transmission/replication in SANs. Therefore, *MinCOF* is designed to achieve this goal.

3.4. Framework Design

MinCOF is a coflow based scheduling and routing framework, which is aimed to achieve the following objectives in cloud SANs:

1) **Short Average CCT:** The average CCT of transfer applications should be minimum in order to achieve higher throughput of object I/O storage.

2) **Starvation Free Scheduling:** The coflows in the SANs should not starve for a long time.
3) **Work Conserving Scheduling:** Each application should be allotted a fair share of the resources it requires to finish it's job.
4) **Backward Compatibility:** The framework should not affect other applications which do not use the functionality of *MinCOF*.
5) **Proprietary System Avoidance:** The framework should not depend on any proprietary system tools or licensing costs, and mostly be open source and easy to use.
6) **Immediate Deployablity:** The framework should incur minimal customization on the built-in commodity hardwares so that it can be deployed easily across different varieties of OS platform.

The designs which achieves all these features are further explained in the subsequent subsections.

3.4.1. Framework Workflow

In this section, an overview of the different framework components of *MinCOF* is described using the Figure 3.1.

The framework initialization steps are

1) Each link from a leaf layer to spine is sliced into three queues for prioritizing three different kinds of coflows, namely, Scheduled, Starved and Best Effort (Figure 3.1(d)). (section 3.4.2)
2) Coflow aware-transfer application is initialized on each host (Figure 3.1(c.c1) and (c.c2)). (section 3.4.3)

   The initialization steps for each coflow are

   1) Each flow in a transfer application is assigned a unique coflow ID on the hosts. The transfer applications report the per flow bandwidth information in coflows to the coflow scheduler (Figure 3.1 (c.c1) and (a.c1)). (section 3.4.3)
   2) The scheduler then schedules the coflows. (section 3.4.4)

   The repetative steps during the live time of each coflow are

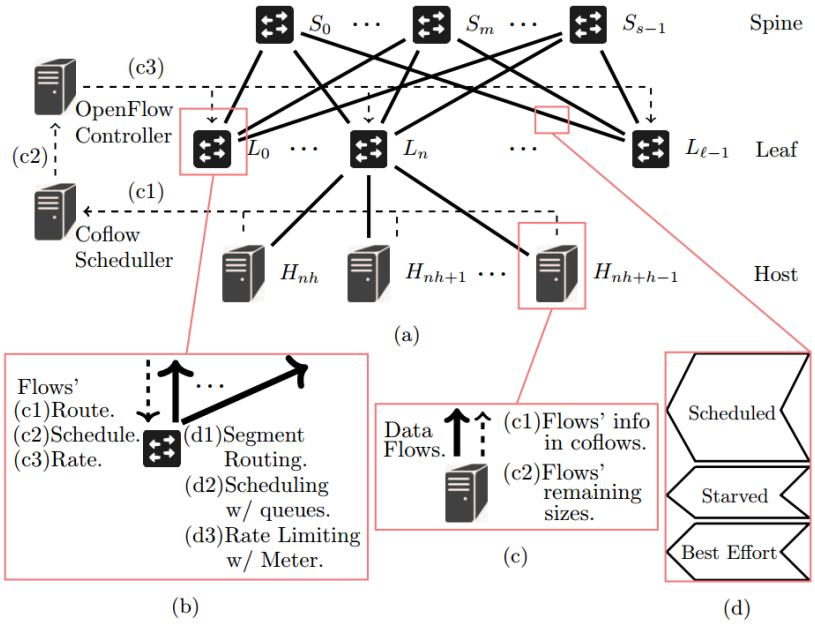


Figure 3.1. (a) The overall architecture of *MinCOF*. Dashed links carry control messages. To save space, hosts connecting to leaf switches other than Ln are omitted. (b) The functions on the leaf OpenFlow Switches. (c) The functions on the hosts. (d) The slices in links from leaf OpenFlow Switches to spines for prioritizing coflows.

1) The transfer applications dynamically update the remaining data size of each flow to the scheduler (Figure 3.1(c.c2) and (a.c1)). (section 3.4.3)
2) Whenever a new coflow arrives or a coflow finishes, the scheduler recalculates the scheduling of the coflows. It then sends the coflow scheduling, coflow routing and per flow rate limiting information to the OpenFlow Controller (Figure 3.1(a.c2)).
3) The controller then assigns the per-flow based scheduling and routing rules on the OpenFlow Switches(Figure 3.1(a.c3) and (b)).

3.4.2. OpenFlow 1.3 Switch

All leaf switch in *MinCOF* uses OpenFlow switches for easily deploying the rules from the controller and let the controller and the scheduler handle the framework overhead. Each link from the leaf switch to a spine is divided into three queues, Scheduled queue (50% of port capacity), Starved queue (20%) and Best Effort queue (30%) as shown in Figure 3.1(c). When the coflow scheduler schedules the coflows, then the coflows with exclusive bandwidth for

it's flows to use, and the rate limiting configuration of the flows are assigned to the Scheduled queue. All remaining coflows are assigned to the Starved queue. This is **to ensure that no coflows starve** while waiting for it's turn in the coflow scheduling order. If some coflows requires more traffic capacity than the Scheduler or Starved queue, then **to ensure the work conservation**, they are assigned to the Best Effort queue. The Best Effort queue and the Starved queue compete freely for the bandwidth. However, a fair share among the flows can easily be deployed using the active queuing disciplines such as FaLL [57]. The best effort queue length is set to 5% Bandwidth-delay Product (BDP) to keep queuing delay low. **To support backward compatibility**, flows which do not have coflow attributes are placed in the best effort queue.

The coflow routing is achieved by adopting the MPLS segment routing (SR). Each scheduled flow is appended with an MPLS label assigned by the coflow scheduler when first reaching a leaf switch. This MPLS label is used to track the path through which a flow should pass between source-destination pairs. To support routing, any conventional flow needs to match the 5-tuple <Protocol, Src. IP Addr., Dst. IP Addr., Src. Port, Dst. Port>. It requires a computing overhead on the switch to parse each packets. However, the benefit of using MPLS SR is the simplicity, where the next route can easily be known through the MPLS label. Spine switches can be regular switches which are simply forwarding packets according to the MPLS label.

For per-flow rate-limiting, *MinCOF* utilizes the OpenFlow Meter. The coflow scheduler passes the corresponding rate limit of each flow in scheduled coflows in every new coflow schedule. The rate limit is configured on the OpenFlow Meter table to control the transmission speed of flows.

3.4.3. Coflow-aware Transfer Application

A typical transfer application consists of the following steps:

1) Command line argument is passed to start the transfer application.
2) Multiple parallel TCP flows are created and data is evenly distributed across the flows for transmission.
3) A server periodically accumulates the overall data transmitted and reports the statistics

of the transmission progress.

To provide proprietary free solution, a coflow-aware transfer application is designed, by incorporating a regular transfer flow to send its coflow information to the coflow scheduler. Henceforth, no proprietary monitoring tool is needed. A 1-way coflow information is passed to the scheduler from each hosts. This benefits in simplicity over the existing 2-way coflow synchronization mechanism, which requires a server to accept all the incoming traffic on each host. As a result, rigorous security and system management policy configuration requires to be modified. **To avoid customization and ensure easily deployability**, the *MinCOF* protocol only requires UDP socket.

The coflow-aware transfer application workflow consists of the following steps:

1.1) Command line argument is passed to start the transfer application.

1.2) Each flow is assigned a unique coflow ID, and the coflow scheduler location.

2.1) Multiple parallel TCP flows are created and data is evenly distributed across the flows for transmission.

2.2) The coflow information is synchronized with the coflow scheduler using the 1-way egress UDP segment.

3.1) A server periodically accumulates the overall data transmitted and reports the statistics of the transmission progress.

3.2) Step 2.2 is repeated and the coflow information is again synchronized dynamically to the coflow scheduler.

An example integrated transfer application is the customized BBCP of the BIC-LSU big data storage area network [6].

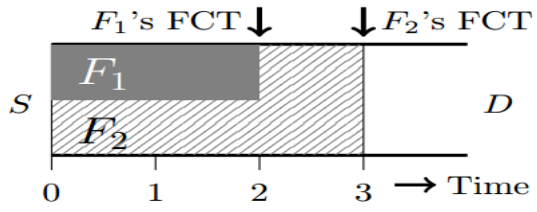
3.4.4. Coflow Scheduler and OpenFlow Controller

The main components of *MinCOF* is the coflow scheduler and the OpenFlow Controller. The coflow scheduler is responsible for making three important decisions: the per-flow rate limiting, coflow scheduling and coflow routing, based on the coflow synchronization infor-

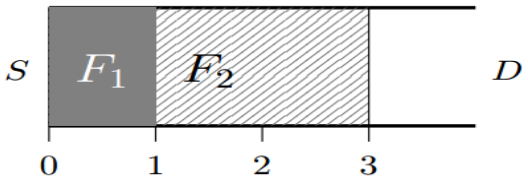
mation from the hosts during the execution of the coflow-aware transfer application. The controller on the other hand, enforces the network allocation decision made by the scheduler to the corresponding OpenFlow switches.

In order to show how the three decisions taken by the coflow scheduler, help in achieving better average CCT, a comprehensive example is illustrated in Figure 3.2 by applying the shortest job first heuristic. All paths in examples have the bandwidth $\frac{1}{\text{unit time}}$ and all flows/coflows arrive at Time 0. **First**, we introduce the per-flow rate-limiting. In Figure 3.2(a), we first consider two flows, F_1(Size 1) and F_2(Size 2), which shares one link without per-flow rate-limiting feature. The average flow completion time (FCT) is 2.5. The optimal schedule is achieved in Figure 3.2(b), when flow F_1 is given higher flow rate than the flow F_2. **Second**, we explain the coflow scheduling. In Figure 3.2(c), two coflows, C_1 with Flow $C_{1,1}$(Size 1) and Flow $C_{1,2}$(Size 1) and C_2 with Flow $C_{2,1}$(Size 2) and Flow $C_{2,2}$(Size 1) share two independent links. The Coflow scheduling schedules all flows in one coflow as a logical unit. The optimal schedule is achieved when coflow C_1 is given first scheduling preference over the C_2 on both sources. in Figure 3.2(d). **Third**, we elaborate the effectiveness of coflow routing. In Figure 3.2(e), two coflows, C_1 with Flow $C_{1,1}$(Size 3) and Flow $C_{1,2}$(Size 1) and C_2 with Flow $C_{2,1}$(Size 2) and Flow $C_{2,2}$(Size 3) share two alternative paths. Coflow routing finds the best route all flows in one coflow as a logical unit. The optimal routing decision is achieved in Figure 3.2(f).

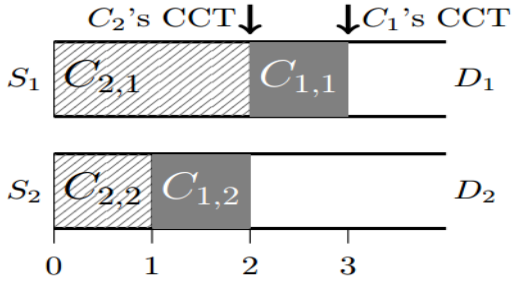
Inorder to avoid using proprietary systems, the coflow scheduler maintains a Data Structure, $Path$. The entries of the table stores the bandwidth usage of the coflows in the Scheduled queue in each leaf-to-leaf switch. The per-flow rate-limiting ensures that the usage information in Path is precise. Therefore, the total number of entries in the table will be equal to the total path in the SAN. In Figure 3.1, the total path tracked for computation is $s * l$, where s and l is the total number of spine and leaf switch respectively. As the blocking ratio (the ratio of the total number of upward links to downward links) in SAN is typically high (10:1), the total table entries in $Path$ is usually limited and can efficiently be implemented using the hash table, for faster insertion and deletion. Function PathRem(i, j, p, h) returns the remaining available



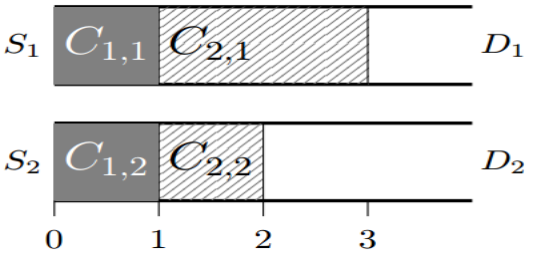
(a) 2 flows must fairly share one path without per-flow rate-limiting. Avg. FCT = (2+3)/2 = 2.5.



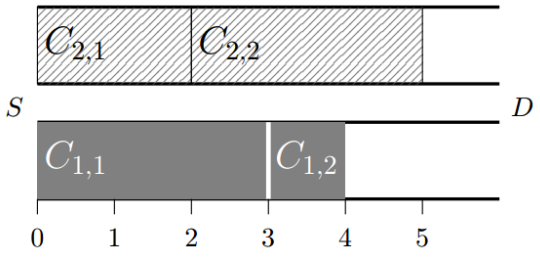
(b) 2 flows optimally share one path with per-flow rate-limiting. Flows can transfer in arbitrary order. Avg. FCT = (1+3)/2 = 2.



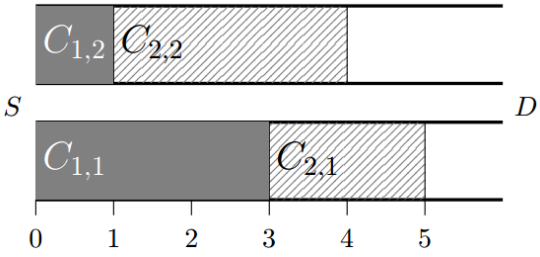
(c) Random schedule of 2 coflows. Avg. CCT = (2+3)/2 = 2.5.



(d) Optimal coflow schedule. Avg. CCT = (1+3)/2 = 2.



(e) Random routing of 2 coflows via 2 alternative paths. Avg.CCT = (4+5)/2 = 4.5.



(f) Optimal coflow routing. Avg. CCT = (3+5)/2 = 4.

Figure 3.2. (a)(b) Impact of coflow scheduling. (c)(d) Impact of per-flow rate-limiting. (e)(f) Impact of coflow routing. Each path has bandwidth $\frac{1}{\text{unit time}}$. The three features enable coflow scheduler to apply the shortest job first heuristic and its variations.

bandwidth from Host i to Host j via Path p.

Our coflow scheduler is able to estimate the capacity gain of re-routing a TCP flow. We assume that every host uses the default Ethernet Maximum Transmission Unit (MTU) size 1500 bytes. Due to the stably short queue at each switch port (section 3.4.2), BDP can be estimated, so can the TCP's congestion window size (Cwnd) using the theoretical model in [36], Cwnd BDP/MTU. The duration of a re-routed flow is reasonably assumed to be the inter-coflow arrival time. Figure 3.3 illustrates the worst impact of re-routing. The area below the curve is the

effective capacity (BDP) of the flow. $C_O(C_N)$ is the effective capacity on the old(new) after re-routing and is calculated in Eq. 3.2(Eq. 3.4) respectively. The overall capacity gain of rerouting a TCP flow to a faster path is calculated by $C_N - C_O$.

Whenever a new coflow arrives/finishes, SERVECOFLOW is invoked and a new coflow schedule is generated for all coflows as shown in Algorithm 5. The newly arrived coflows first are routed through the DCN for the load balancing (Line 17-23). The average CCT bottleneck is measured based on the time taken by the slowest flow to complete using Eq. 3.1. If there is another route available with smallest CCT for the slowest flow based on the bandwidth estimation discussed in the previous paragraph, then the scheduler would schedule the slowest flow through that new route (Line 2-5) **to shorten the average CCT**. The coflows are scheduled based on the shortest job first heuristic (Line 6-12), and the coflows are allotted exclusive bandwidth in the Scheduled queue (C_{sch}) (Line 25-30) to share large portion of the link bandwidth. The other coflows are moved to the Starved queue (C_{str}) **to prevent starvation** (Line 31-35).

$$\Gamma = \max_i \left(\max_j \left(\max_p \frac{d_{ij}}{PathRem(i,j,p,h)} \right) \right) \tag{3.1}$$

where d_{ij} is the summation of the remaining data size of the current coflow from Host i to Host j.

$$C_O = D\frac{W_O}{2} + \left(D \textbf{ div } \frac{W_O}{2} \right) \frac{W_O^2}{4} + \frac{\left(D \textbf{ mod } \frac{W_O}{2} \right)^2}{2} \tag{3.2}$$

$$D_h = D - \frac{W_N}{2} \tag{3.3}$$

$$C_N = \frac{W_N^2}{4} + D_h\frac{W_N}{2} + \left(D_h \textbf{ div } \frac{W_N}{2} \right) \frac{W_N^2}{4} + \frac{\left(D_h \textbf{ mod } \frac{W_N}{2} \right)^2}{2} \tag{3.4}$$

where **div** calculates the quotient of integer division and **mod** calculates the remainder of integer division.

Algorithm 5 Coflow Scheduling & Routing in OpenFlow-baseed spine-leaf DCN

```
1: function SCHEDULE(Coflows γ, PathRem(.))
2:     C_sch = Sort all Coflows in γ in SBF order
3:     if Rebalancing shortens 1st coflow's CCT ∈ C_sch then      ▷ Measured using Eq. 3.1, Eq. 3.2, Eq. 3.4
4:         Rebalance flows ∈ 1st coflow
5:     end if
6:     for all coflow C in C_sch do                                ▷ allocate BW
7:         Calculate Γ using Eq. 3.1
8:         for all flow in C do
9:             rate ← (flow's remaining size)/Γ
10:            Update PathRem(src, host, dst, p, h) with rate
11:        end for
12:    end for
13:    C_str = starved coflows in C_sch
14:    return C_sch, C_str
15: end function

16: procedure SERVECOFLOW(Coflows γ, PathRem(.))
17:    if new coflow arrival then
18:        for all flow ∈ new coflow do
19:            if flow traverses across leaf SWs then
20:                Place flow on least used inter-leaf path
21:            end if
22:        end for
23:    end if
24:    C_sch, C_str = SCHEDULE(γ, PathRem(.))
25:    for all flow in C_sch do
26:                                          ▷ place scheduled flows on src leaf SWs
27:        Configure new segment routing path
28:        Migrate flow to Scheduled queue at egress port
29:        Configure flow's OpenFlow Meter entry
30:    end for
31:    for all flow in C_str do
32:                                          ▷ place starved flows on src leaf SWs
33:        Migrate flow to Starved queue at egress port
34:        Remove flow's OpenFlow Meter entry
35:    end for
36: end procedure
```

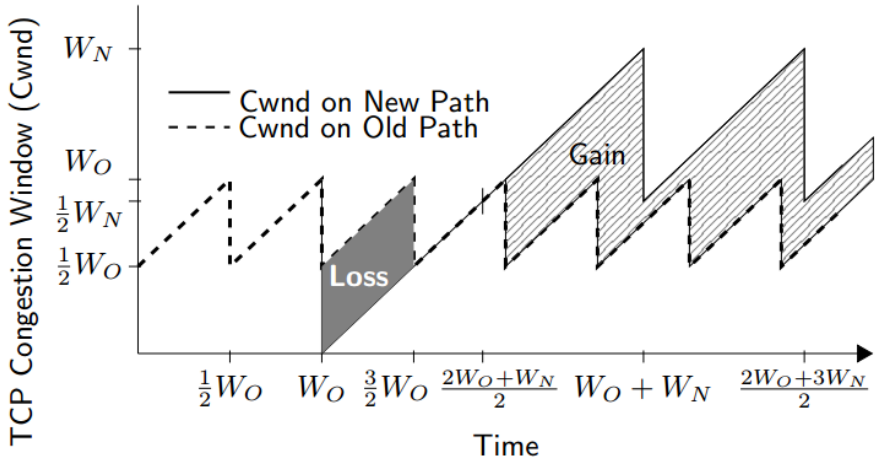


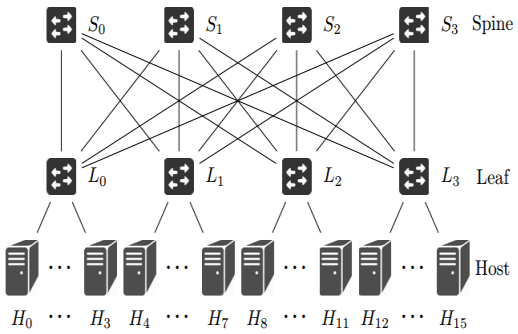
Figure 3.3. The worst impact of switching paths on the congestion window size of a TCP flow. The flow re-routes to a faster path at Time W_O. Transmission capacity loss because of TCP re-ordering (shaded area) and gain (slashed area) are marked.

3.5. Evaluation

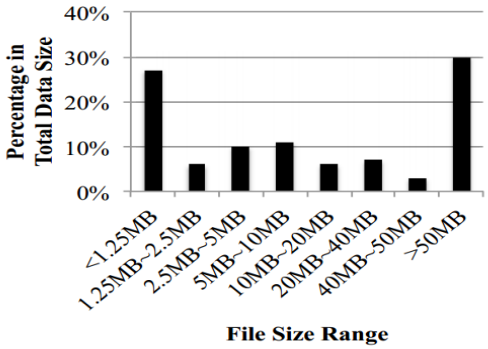
MinCOF is compared with respect to other state-of-the-art coflow scheduling/routing framework such as Varys with ECMP routing, and Tailor on a DCN testbed with synthesized SAN workloads. Rapier is not considered for this study due to its complexity in implementation. Each result is the average of 5 runs and normalized to the result of Varys.

3.5.1. Testbed Environment

The fat-tree topology with 4 spine switches, 4 leaf switches and 16 hosts are considered in the DCN as shown in the Figure 3.4(a). The commodity servers has been used to emulate the hosts and switch for the testbed environment. Each switch and host has 8 Intel Xeon 2.33GHz CPU cores, 8GB main memory. Each switch has 2 Intel I350-T4 network interface cards. All servers run the Ubuntu Linux 16.04. Each leaf switch is deployed with Open vSwitch(OVS) OpenFlow Switch [19]. The coflow scheduler is written in Python, and the OpenFlow Controller is built on the Rye framework [12]. However, the OpenFlow Meter is not implemented on OVS. A primitive control framework is created for the OpenFlow Controller to manipulate the tc command on each switch to mimic the behavior of the OpenFlow Meter using the Linux kernel packet scheduler. All network interfaces are throttled to 100Mb/s to ensure that



(a) The spine-leaf fat-tree experiment DCN consists of 4 spines, 4 leaves, and 16 hosts. Each link is 100Mb/s.



(b) The downscaled file size distribution of Data Oasis.

Figure 3.4. Simulation Environment.

Table 3.2. Statistics of coflow inter-arrival time in the Facebook traffic trace.

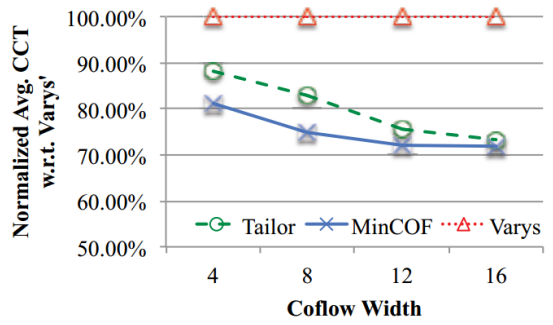
Mean	Median	Mode	Std. Dev.
6.91s	3.06s	1.58s	9.76s

the processing power of the CPUs on the switch is competent for forwarding network traffic. The RTT at a host is around 350μs.

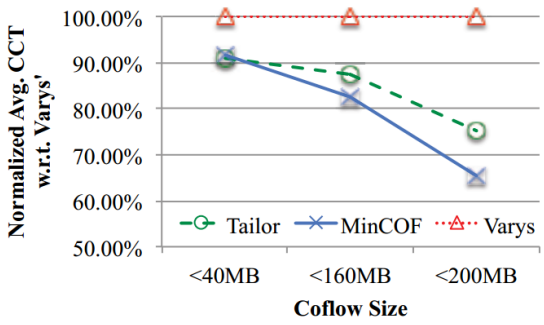
3.5.2. Workload

Due to the scarcity of wide variety of workload distribution in cloud storage environment, a scientific big data size distribution in the Data Oasis storage cluster is considered for the traffic generation [33].The actual file size in the dataset is scaled down by multiplying the testbed bandwidth to Data Oasis' SAN bandwidth ratio, $\frac{\text{Exp. Net BW}}{\text{Data Oasis Net BW}} = \frac{100\text{Mb/s}}{10\text{Gb/s}} = 0.01$. The scaled file size distribution is summarized in Figure 3.4(b). To increase scalability, *MinCOF* schedules only at coflow arrival/finishing and only considers the coflows carrying the large files ($> 2.5MB$) which already account for around 70% of total data size. Consequently, 64 coflows with the arrival times distribution from the Facebook trace in [11] is generated. Each coflow involves 3 through 12 normally distributed host-to-host file transmissions. Each file transmission uses 4 through 16 flows.

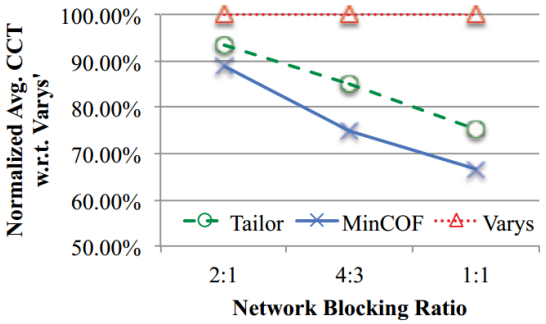
3.5.3. Impact of Coflow Width



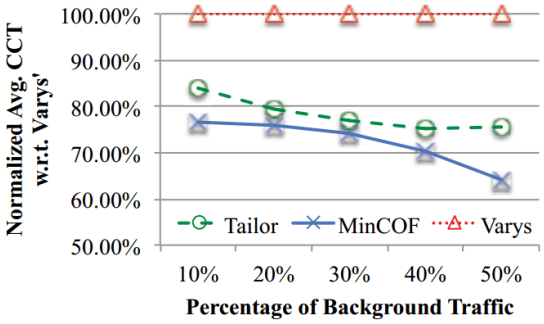
(a) Impact of Coflow Width. Shorter CCT is better.



(b) Impact of Coflow Size. Shorter CCT is better.



(c) Impact of Network Blocking Ratio. Shorter CCT is better.



(d) Performance under Background Traffic. Shorter CCT is better.

Figure 3.5. Comparison between Tailor, and *MinCOF* normalized w.r.t Varys with different parameters.

A coflow width is defined as the number of flows in coflows. Coflows with only 4, 8, 12 or 16 flows are generated. The average CCT is evaluated under various coflow width as shown in Figure 3.5(a). As the number of flows increases, each flow becomes shorter and finishes faster. *MinCOF* and Tailor performs better than Varys, due to its coflow routing feature added to its framework. *MinCOF* further improves by at most 9.81% compared to Tailor in the case of 8 flows. The saturation with larger number of flows is due to the naive implementation of the OpenFlow Meter. A deployment on dedicated hardware OpenFlow Swtich would alleviate such saturation as the OpenFlow protocol message format and message passing mechanism is standarized and highly optimized.

3.5.4. Impact of Coflow Size

MinCOF is further evaluated with the varying coflow size, the total size of all associated flows in the experiment. The file size range is divided into three categories, $< 40MB (4 \times 10MB), < 160MB (4 \times 40MB), and < 200MB (4 \times 50MB)$. The last range ends at 50MB because of the scaled down file size upper upper limit in the OpenStack Swift cloud object storage is 5GB. Figure 3.5(b) presents the result. *MinCOF* demonstrates more advantage with larger coflow size, up to 12.94% compared to Tailor, if the coflow can be as large as 200MB. The overhead problem mentioned in section 3.5.3, again, results in the saturation between *MinCOF* and Tailor in the case of small coflow.

3.5.5. Impact of Network Blocking Ratio

In this experiment, the performance of coflow scheduling/routing frameworks in the spine-leaf fat-tree DCN with various blocking ratios is observed. A higher blocking ratio reflects that communications between hosts connecting to different leaf switches are more difficult. The result is summarized in Figure 3.5(c). With high blocking ratio, the severe congestion in the network lessens the improvement of routing so that three frameworks have similar performance. With low blocking ratio, *MinCOF* effectively routes flows among multiple available paths so that achieves the largest improvement of 11.33% compared to Tailor.

3.5.6. Impact of Background Traffic

To disable the feature of coflow scheduling/routing in order to study the backward compatibility, randomly selected coflows are blocked to send their coflow information to the scheduler. Background traffic in practice is from applications which are not yet integrated with the coflow scheduling/routing framework. The percentage of background traffic gradually increases from 10% through 50%. The result is shown in Figure 3.5(d). Tailor and *MinCOF* outperform Varys because of their routing feature. However, Tailor saturates with 40% background traffic. *MinCOF* profits from the sliced leaf-to-spine links (section 3.4.2) which confine the bandwidth consumed by background traffic and still improves under heavy background

traffic. The maximum improvement is 15.21% with 50% background traffic.

3.6. Conclusion

Existing coflow scheduling/routing framework requires customized OSs on hosts or proprietary network monitory components to achieve faster application completion time. However, such framework adds deployability issues and/or extra costs in SANs. *MinCOF* inherits all the features of coflow scheduling, routing and rate limiting, along with minimal modifications to applications. It also do not require any proprietary components due to the use of commodity OpenFlow Switches, and open source software. Experiment results shows that *MinCOF* shortens the CCT by upto 12.94% compared to latest OpenFlow based framework, Tailor. *MinCOF* is further implemented to the BIC-LSU 10/40 Gb/s OpenFlow based spine-leaf big data SAN.

Chapter 4. Identify Fake News During Natural Disasters

4.1. Introduction

Social media platforms have become an integral part of the life styles of many individuals since such platforms allow easy access to diverse information and rapid interactions among their users. Unequivocally, the growing impact of fake news has been increasingly affecting the integrity of the society in a harmful way. Therefore, detecting fake news and understanding its characteristics and mechanisms of dissemination constitutes the community-wide responsibility for the stability and the sustainability of the society in a modern and open internet ecosystem.

In spite of many prior studies conducted to detect fake news primarily relied upon social media contents, due to the non-trivial task of semantic modeling dealing with both misinformation and disinformation [43], a successful strategy is highly likely to be achieved only when we understand the complexity on how they are engaged and disseminated [14,42,47,53]. Consequently, it is a holistic approach that takes into account multiple characteristics of fake news namely, user engagement, user profile, and the dissemination of entire fake news event, along with the fake news texts or the content-based information [44]. In this work, we introduce an end-to-end solution by leveraging recent advances in Deep Learning, which is intrinsically suitable for combining diverse sets of features without hand-crafted feature engineering and it is also inherently scalable for large datasets [44].

In order to address the complexity of requirements for an effective solution to deal with malignant fake news spread in the time of disasters, we introduce a pipeline framework for detecting fake news events which delivers false information to a large population quickly and broadly on Twitter. It is worth mentioning that our framework is capable of dealing with other types of fake news events (i.e., non-disaster domains) as well. Our framework is streamlined with the Twitter data collection component and the Deep Learning-based data analysis com-

D.K. Singh, S. Shams, J. Kim, S. Park, and S. Yang, "Fighting for Information Credibility: An End-to-End Framework to Identify Fake News during Natural Disasters", 2020 Information Systems for Crisis Response and Management (ISCRAM)"

ponent. Notably, the data analysis, which models a classification task, incorporating multiple features (such as user features) as input along with the temporal sequence of fake news propagation. This analysis is designed to be effective in cases with an insufficient size of the training dataset, which is commonly faced during a specific disaster event.

In the following section, we introduce several studies which are most relevant to our study, followed by the methodology, experiments, and the results and discussion sections. Finally, we conclude with a brief summary of our study and plans for future work.

4.2. Related Studies

Pierri and Ceri (2019) present a comprehensive survey of publications on recent advances in fake news studies in three categories: the detection of false news; the characterization of fake news spreading in social media; and the mitigation approaches to alleviate the impact of such news on communities [39]. For research contributions on detecting false news, the authors further group the publications into three categories based on the features analyzed: the content of fake news, (social) context of fake news which propagate in social media, and combination of both content and context. Content-based approaches identify linguistic features from the textual content of fake news [27, 28, 38, 40, 41, 54]. Manually-engineered feature sets are often used along with traditional machine learning and deep neural network models. One of the early studies for fake news detection, which focused on classifying the relative stances of news content to its title, also goes into this category [25]. Social context-based approaches aim to distinguish fake news cascades (which are a group of social media posts sharing the same fake news) by analyzing the social aspect of the fake news which diffuse in social media [32, 49, 52, 55, 56]. The profiles of users involved in fake news activities, interactions between users, and interactions between users and fake news are all important features to consider in identifying fake news cascades. Context-based approaches may also analyze the spatiotemporal features (e.g., time stamps and geolocation data of Twitter posts) as long as these features are provided as part of the dataset. Finally, the authors introduce the combined approach, which mixes both content-based and social context-based approaches, as

the latest and most effective methodology for fake news detection [44, 47, 51].

Considering that the combined approach is the latest and the most effective one, we have also based our proposed detection model on this approach. In the following paragraphs, we further introduce selected studies, which have incorporated the combined approach in their methodologies. These studies use datasets which have multiple dimensions (e.g., texts, user profiles/interactions, spatiotemporal data), and also apply deep neural network models (e.g., Convolutional Neural Net, Recurrent Neural Net) to detect fake news and to facilitate the study of fake news detection in social media.

Ma et al. (2016) developed an approach which converts Twitter data into variable-length time series and then trains four recurrent neural network (RNN)-based models (tanh-RNN, long short-term memory (LSTM) [22, 26], gated recurrent unit (GRU) [7] with 1-layer hidden units, and GRU with an extra hidden layer) for rumor detection problem [34]. Deep neural network models, including RNN models [45], demonstrated advantages over traditional machine learning models. The authors shared their own training datasets after developing them by using both manual and automatic processes based on Twitter and Weibo (Chinese social media) platforms. Their algorithm converts incoming microblog streams into a variable length time series data. This approach captures and represents the densely populated regions in the diffusion stream for analysis. The model performances were compared to those of traditional machine learning models, and it showed outstanding performance for the Weibo dataset in terms of the accuracy, precision, recall, and F measures. Although their GRU models performed well for the Twitter data in terms of the accuracy and F measure, traditional support vector machine (SVM) models showed slightly better performance for precision and recall measures. Our study also adopts an RNN model and use Ma et al.'s Twitter dataset (i.e., Rumdect dataset) as part of the training model. However, the difference is that our model includes a user module component to address the social aspect of the fake news propagation. In addition, our model is further fine-tuned using our own disaster fake news Twitter dataset applying the transfer learning (TL) scheme.

The hybrid model for detecting fake news diffusion in social media, presented by Ruchansky et al. (2017) [44], focuses on three characteristics of fake news: textual content of the fake news, user responses the fake news receives, and the sources of the fake news (e.g., structure of URLs, publisher, users). Their model consists of three components, namely Capture, Score, and Integrate. The textual content of fake news and temporal aspect of user-to-news interactions are analyzed by the Capture module using a paragraph embedding technique (e.g., doc2vec) [31] and the volume of fake news posts per unit time, respectively. In order to uncover the source characteristics, their Score module operates on all users and then depicts frequencies of the user-to-user engagement mediated by fake news articles. Finally, the results from both Capture and Score modules are concatenated in the Integrate module for predicting whether each article is fake or real. Ruchansky et al. adopts the same social media data (Twitter and Weibo) collected by Ma et al. However, their approaches for preparing training datasets are not identical. For Ma et al., they produce variable-length time series data for training their models out of the collected social media data whereas Ruchansky et al. partition the social media data into 1-hour segments and then each of which becomes an input to a cell for more efficient training of the model.

One of potential limitations of Ruchansky et al.'s model is that their model depends heavily on user-to-user interactions in the Score module, which presides over all users in the training dataset [44]. This design consideration may make the entire model less flexible and perform less effectively when the model has to predict labels for an unseen set of users and articles with which those users interacted. Our model design is similar to that of Ruchansky et al.'s in that our model has the Content Aware module, which corresponds to their Capture module. Our model also has the Context Aware module, which corresponds to their Score module. However, the differences are that: (1) we attempt to resolve the issue of their model's dependency on user-to-user interactions by representing users with their Twitter profile data; and (2) we adopt the TL scheme to overcome the insufficient size of our training dataset, which is derived from fake news events (tweets) in disaster domain. We hope that this study will help

the emergency organizations, institutions such as libraries and schools, and the affected public to filter out false information during the critical times of natural disasters. This will increase the community resilience as well.

4.3. Methodology

4.3.1. Data Collection

As part of our framework, we have a distributed tweet collection system infrastructure. We developed this to allow for convenient collection, storage, and filtering of Twitter data. The system collects requested tweets based on keywords and locations of the tweet posts, filters the tweets, and then stores them into the MongoDB database as shown in Figure 4.1.

Ground Truth Label: To collect the ground truth labels for the fake news, we rely on the fact-checking website such as Snopes.com. Snopes.com is a well-regarded evidence based source for filtering out myths and rumors and misinformation on the Internet [48]. Snopes.com classifies a news article into broader spectrum of ratings (e.g., false, mostly false, mixed, mostly true, unproven, miscaptioned). However, we only consider those articles which are labeled as 'false' and 'mostly false' as fake, and 'true' as real for the ground truth labels. We use web automation tool, Selenium [46], to periodically collect the relevant articles to create our ground truth labels.

Social Context: To ensure better search results, the keywords to collect relevant Twitter posts are generated manually based on the headlines of articles in Snopes.com. Related Twitter posts are then collected by using the Search and Streaming APIs, provided by the Twitter's Advanced Search API. Twitter's Streaming API helps us to define a bounding box of geolocations for the tweets that need to be collected, whereas Search API allows the hashtag- or keyword-based collection of tweets. We utilize both of these features to collect the disaster-related tweets. We also used the Twitter's Search box to search and collect the fake news-related tweets explicitly. Once the tweets are collected, we apply de-duplication and filtering processes to remove duplicate tweets that will inevitably be collected from both types of APIs and filter the noise (irrelevant tweets) respectively, based on the text, time interval, and geolo-

cation to study our fake news spatio-temporal patterns. After we obtain social media tweets, which are directly relevant to the fake news, we further fetch metadata such as user profiles, replies, likes, retweets, and social network information from those tweets to create our training dataset to train our proposed detection model described in Detection Model section.4.3.3

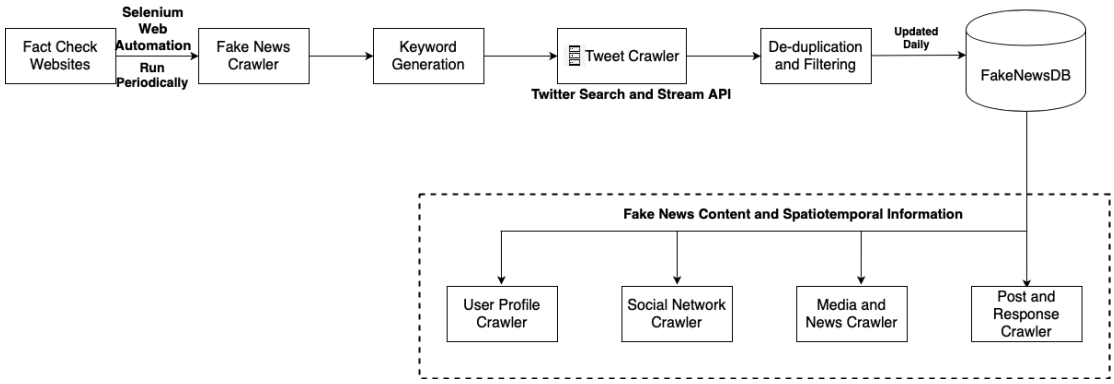


Figure 4.1. Twitter Data Collection System Infrastructure.

4.3.2. Data Preparation

Our model is comprised of two modules-Content Aware and Context Aware modules and the input data is prepared accordingly. Following the model proposed by Ruchansky et al. [44], the input data to our Content Aware module is generated. Each Twitter event e is considered as a temporal sequence of sets of tweets within t time interval. Each engagement between a user u_i and an event e_i at time t is represented as a tuple of $X = (\epsilon, \Delta T, X_u, X_t)$, where ϵ is the number of tweets engaged in a particular time interval ΔT. X_u is the row in the adjacency matrix between the event e_i and user u_i involved in ΔT, and X_t is the numeric vector representation of the tweet texts within ΔT. For example, let there be $e_1, e_2, ..., e_n$ events. Each event e_i is comprised of a series of tweets $t_1, t_2, ..., t_k$ by different users $u_1, u_2, ..., u_l$. Furthermore, we partition the series of tweets into different sets $s_1, s_2, ..., s_m$ based on ΔT time interval. Therefore, X is the vector of tuples $(\epsilon, \Delta T, X_u, X_t)_1, (\epsilon, \Delta T, X_u, X_t)_2, ..., (\epsilon, \Delta T, X_u, X_t)_m$, where ϵ is the total number of tweets in a set s_i. To calculate X_u, we first create an adjacency matrix of all the users and events, where each row represents the number of times a user u_i involved in each event e_1 to e_n. Moreover, we take the mean of the collection of rows (or users involved in a

set s_i) of the adjacency matrix and represent it as X_u. In the end, we reduce the dimensionality of the adjacency matrix using PCA (with dimension 20). Similarly, in order to calculate X_t, we consider the collection of tweets in a set s_i and then use doc2vec to produce the vector embedding.

For the Context Aware module, various metadata from the Twitter user profiles are collected based on those twitter users involvement in a particular event e_i using our data collection system described in Data Collection section 4.3.1. Each user's number of tweets shared, number of followers, number of following, number of likes, the links or media shared, the age of their profile, if their account is private or verified, and their geocode locations are all taken into account. The user profile collected and processed in this way may capture user characteristics based on their activities in the network. Thus, our Context Aware module should be able to distinguish users who may have different levels of susceptibility for fake news. In Ruchansky et al.'s study, user features are constructed based on the user-to-user interactions within the training dataset. Such features are dependent on the dataset and if the test set encounters new unseen users involved in the fake news, then such model would fail to capture new fake users.

4.3.3. Detection Model

In this section, we go over the deep learning model utilized for disaster-related fake news detection. Any news article contains three important features: the information an article carries (e.g., tweet text), the response to the article by various clients engaged with that article (e.g., replies to the tweet, the number of likes, the number of followers, and the temporal information), and the source of the article (e.g., twitter account which posts that tweet and it's behavior on social site). Therefore, by incorporating these three features in our deep learning model, our model could learn the underlying data distributions for fake news behaviors in social media in a more comprehensive way to distinguish between a real and fake news. Inspired by Ruchansky et al. [44], we constructed the content aware module to capture the temporal and textual fake news information. However, we added a context aware module to capture

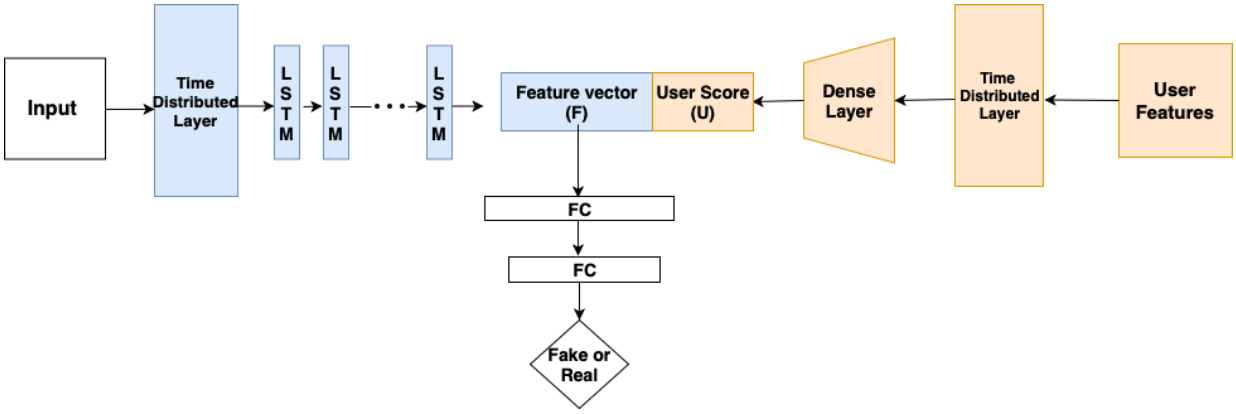


Figure 4.2. Model Detection Specification.

user features based on the individual twitter user profile. As previously mentioned, the main drawback of Ruchansky et al.'s model is the dependency on user-to-user interactions in the Score module. This design consideration may make the entire model less flexible and perform less effectively when the model has to predict labels for an unseen set of users and/or articles. We tackled this limitation by designing the content aware module to characterize each user's information based on it's profile. The second implementation constraint that we handled is the fact that the disaster-related dataset is relatively small in comparison with fake news dataset used in previous works. Therefore, we utilized TL to overcome this limitation. We will go over the implementation in detail in the rest of this section.

Content Aware Module: The goal of this module is to capture the temporal and textual features of the fake/real events. We incorporated a time distributed embedding layer to standardize the input X before feeding it to the RNN. To capture the temporal characteristics, we used the Long Short-Term Memory (LSTM) model, since LSTM models are good at capturing long-term dependencies and can also handle variable length inputs. The final hidden layer of LSTM is then passed through a fully connected layer to get a low-dimensional feature vector F which represents the temporal and textual characteristics as shown in Figure 4.2.

Context Aware Module: This module captures the user features based on the Twitter user profile information. The user feature matrix is constructed based on their user profile and then Principal Component Analysis (PCA) is applied. PCA not only helps in reducing the di-

mensionality, but also enables us to capture the most important variables in the original feature space. Therefore, the dimension of the user feature matrix is reduced from eight to four (based on the elbow in the curve as a function between the dimensions and the explained variance, with no rotation). Given the set of low-dimensional user features (of dimension 4), we then apply dense layers to extract the user feature vector U and apply masking to select only those users that are involved in a particular batch. The output of both the modules are concatenated and the whole model is trained jointly.

Training with Transfer Learning: In the disaster domain we face the limitation of the sufficient amount of data, due to the limited time window of the disaster event. TL is a traditional novel approach to handle training on small datasets and achieve a robust model, given we have similar larger data distribution available from different domains. Thus, in our TL, we have pre-trained our model using a bigger dataset and then transfer the learning on a specific (but relatively smaller) natural disaster [37]. Therefore, the pre-trained model have the same architecture and was trained with an existing larger fake news event dataset (Rumdect). In this first step, a rich set of generic features were extracted from the data in diverse topical areas. Then, the fine-tuning with our disaster data allows our model to be trained in spite of the smaller disaster dataset (Table 4.1).

Note that the TL scheme is only applied to the Content Aware module. The Content Aware module is then jointly trained with Context Aware model for the end-to-end supervised task. The Loss function for training our model is composed of the cross entropy loss ($\mathscr{L}_{\text{ce}}$) and the L_2 regularization as follows.

$$\mathscr{L} = \mathscr{L}_{\text{ce}} + \lambda \| W \|_2^2$$

The weights for Dense Layer and Time distributed layers are randomly dropped out for training to reduce overfitting. Therefore, we have created an end-to-end framework to collect the fake news during a disaster and then prepared the dataset to feed it to the model. The model is then trained via TL to make predictions. We trained the Context Aware module initially on the larger Rumdect dataset, then the weights of the model were transferred to the

Table 4.1. Overview of the datasets used for training.

	Rumdect Dataset	Disaster Dataset
No. of Events	991	91
No. of Fake Events	497	46
No. of Real Events	494	45
No. of Users	226,791	31,305
Total No. of Tweets	569,912	37,975
Avg. Event Period (hrs)	1,961	3,924
Avg. No. of Tweets/event	575	417
Max No. of tweets	37,475	4,041
Min No. of tweets	4	6
Avg. No. of tweets/user	2	1

natural disaster dataset to handle the small dataset limitations. All the RNN models are trained using Tensorflow 1.8 and tested with Nvidia GeForce GTX 1080Ti. The Rumdect dataset is divided into 80% training, 5% validation and 15% testing dataset, whereas the natural disaster dataset accuracy is measured using the 5-fold cross validation.

4.4. Experiments

4.4.1. Datasets

Since our natural disaster dataset is generated based on the Twitter data, we considered only the Twitter data part from the publicly-available Rumdect dataset for pre-training step as discussed in Transfer Learning section 4.5.2. This dataset originally contained more than 5,000 events with five million relevant tweet IDs [34, 50]. However, when we hydrated those tweet IDs to utilize their full tweet texts and metadata, some of the tweets were found to be no longer existing due to account suspension or deleted posts. Thus, the Rumdect dataset that we hydrated has 991 events and approximately 570,000 relevant tweets. Each event comprises a news story, tweet IDs associated to that news, and the label (real or fake). Upon hydration of the tweet IDs, we get each event with the set of engagements (tweets) made by a user u_i at time t. For our disaster dataset, we have collected 91 instances of fake/real news events along with the profiles of users engaged in the events following the methodology in Data Collection

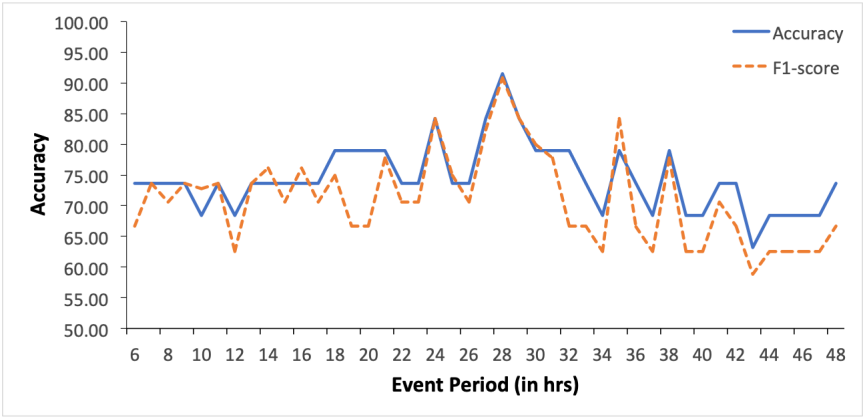


Figure 4.3. Accuracy and F1 score of our model trained with different sizes of event periods.

section 4.3.1. The statistics of both the dataset is listed in Table 4.1.

4.4.2. Model Setup

We partition each engagement of the dataset into segments of ΔT, time interval such that all the engagements within ΔT time are taken as one input to the LSTM cell.

Hyperparameters: We used cross-validation to set the regularization loss parameter to $\lambda = 0.01$, the dropout probability as 0.4, the learning rate to 0.001, and used the Adam optimizer. Furthermore, we considered 1 hour granularity as the partition unit ΔT. For training the doc2vec model, we used the window size of 10 and a vector size of 100. The weights used in the Context and Content Aware modules are of dimension 100.

4.5. Results and Discussion

4.5.1. Fake News Event Detection Performance

In Figure 4.3, the detection accuracies and the F1-score (the harmonic mean of the precision and recall) obtained with our model for different event periods are presented. The different time lengths of event periods from 6 to 48 hours with one hour increment are used and indicated with x-axis labels. Regarding the partition scheme, which is one hour interval, if there is no tweet activity in a certain partition in an event, we skipped it and considered only non-empty partitions in the dataset.

The best accuracy of 91.47 % was achieved when we used the first 28 hours of tweets as an

event period in our disaster dataset. The same condition resulted also in the best F1 score of 90.89. Since in natural disaster domain only classifying the tweets is not enough, rather classifying them within smallest amount of time is also crucial. Therefore, we assessed different time intervals on which we can have the best trade-off between the sensitivity analysis (the overall best accuracy) and the threshold assessment (the minimum number of hours required to achieve the best performance), and found the 28 hours of training to be the right interval which gives us the best accuracy. With this, our model demonstrated the capacity of capturing the textual, temporal, and user engagement/profile aspects of the fake news events in social media successfully.

In spite of reasonably good performance, thanks to TL-based training, the model architecture, and datasets we collected, there exist a challenge to understand the overall results shown in Figure 4.3. For example, more data with a longer event period do not seem to improve the accuracy if the event period becomes longer than 28 hours. There might be two potential reasons for such degradation of the overall model performance: (1) the influx of noisy data and (2) a transition to a more complex data distribution. For (1), it might be related to the fact that noisy and unrelated tweet data were introduced, especially after around 28 hours since the beginning of the fake news occurrences in social media. For example, noisy tweets could have been added to our disaster dataset when social media marketers abused popular hashtags in fake news tweets for their marketing purposes. Another potential explanation might be (2) the transition to different model distribution. After a certain period, different mechanisms to generate fake news events may start to emerge, departing from the data distribution of an early event period (up to 28-hour period) readily detected by the current model. An understanding of whether such mechanisms could turn out to be difficult to model or could be captured by more sophisticated models would be beyond the scope of this work. However, unraveling such a time-dependent behavior we found in this work would be an interesting future work for the nature of disaster fake news detection.

4.5.2. Transfer Learning

As described in Detection Model section 4.3.3, our detection model mainly consisted of two parts: Content Aware and Context Aware modules. The Content Aware portion of the model was trained with TL and the Context Aware module captured the user profile information from social networks that they participated. In order to examine the effects of these different model setup and the role of TL, we computed accuracies using the first 28 hours of event data with different settings.

Combining the Context Aware module along with the Content Aware module had a significant effect on the accuracy with 7.24 % increase in average regardless of the application of TL. This tells us that our Context Aware module could successfully extract useful information from the profiles of users participating in social networks to distinguish false from real information activities. Applying TL to the Content Aware portion also positively affected the accuracy with a 5.3% increase in average regardless of adding the Context Aware module to the detection model. Thus, the issue of insufficient size of training dataset in the disaster domain could be addressed successfully.

If we compare the Content Aware only model without TL (78.94 %) and the Content + Context Aware model with TL (91.47 %), the accuracy difference is much more significant (12.53 %), and it clearly shows the benefits of using the user profile information, as well as TL for the false information detection in disaster domain.

4.5.3. Representing User Characteristics

We calculated the credibility score of each user based on their average participation in the fake or real news events in social media. If a user was involved in fake events only, his or her credibility score became 1. If the user was only involved in real events, his or her credibility score became 0. Hence, the credibility of each individual user was $C_i \in [0,1]$. We considered the 20% random samples of the data and extracted the user features on our trained model. We projected the first two scalar values of the user vector obtained after training, namely feature1

and feature2, with respect to the credibility score of the users. Figure 4.4 shows that the users of high credibility scores (i.e., black and navy dots) were clustered together into two groups located on the top left and on the bottom right. Also the users of low credibility scores (i.e., green and yellow dots), who have a higher tendency to share fake news content, are presented between the two groups of users with high credibility. This clustering result indicated the effectiveness of separating the users involved either in fake or real news events, by considering the features extracted from their user profiles.

Table 4.2. Accuracies for different model setup and with/without TL.

Model Setup	Without TL	With TL
Content Aware Only	78.94	84.21
Content + Context Aware	86.15	91.47

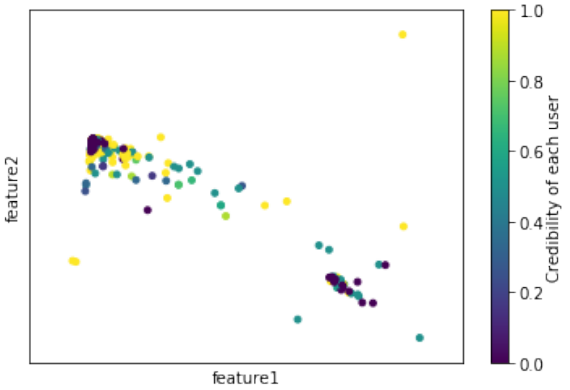


Figure 4.4. User characteristics (Credibility score 0: users involved only in real news events; Credibility score of 1: users involved only in fake news events)

4.5.4. Existing Challenges and Our Contributions for the Disaster Domain

Compared to other domains such as politics, stock market, or celebrities, the size of generated fake news events in disaster domain was relatively smaller. Insufficient size of training datasets in deep learning and machine learning may lead to a creation of an overfitting model. Thus, this insufficient data size was one of main challenges in this study, which we could overcome by applying TL for the Content Aware module in our model (See Table 4.2). Considering that we can make a bigger dataset by collecting fake news events from the forth-coming natural disasters, our model has a potential to be further fine-tuned continuously.

Another challenge was to identify what the optimal length of the event period should be for our dataset to achieve the best performance. We could find that the model trained with the event period length of 28 partitions showed the best performance (See Figure 4.3). Training

with the event period longer than that had an adverse effect on the performance.

Our main contributions for the disaster response domain are two-fold: (1) applying the TL methodology and refining a prior fake news detection model by Ruchansky et al. to address the problem of fake news events occurring during disaster domains and (2) identifying the optimal length of the event period to train and detect fake news events which occur during natural disasters. This may mean that emergency managers performing rescue/recovery efforts could be benefited by identifying devastating false information in social media during disasters within a fairly short time period of 28 hours with a high accuracy (91.47%). Especially, (2) also implies that such early and accurate detection of fake news events could lead to a reduced costs for disaster response (e.g., dispatch rescuers to correct locations and save lives on time), as well as recovery actions (e.g., mobilize food, water, shelter, etc. to the hands of victims and affected on time).

4.6. Conclusion and Future Work

In this study, we aimed to develop a comprehensive framework which was capable of creating a training dataset by collecting and processing fake news-related Twitter data, training an RNN-based detection algorithm, and then making decisions whether a given cascade of Twitter data might be a fake news event or not. Our application domain was natural disasters, and it gave us challenges considering that the number of fake news events occurring during the time of crises, although potentially critical for the safety of the victims and affected, were relatively smaller compared to other domains. To overcome this challenge, we incorporated the TL scheme, which pre-trained our detection algorithm with a larger Rumdect dataset and then fine-tuned the algorithm with our Disaster dataset (See Table 4.1).

We computed the model accuracies and F1 scores by increasing the length of the hourly partitions for the event period (1-hour increments). We could observe that the best performance was achieved when the model was trained with the first 28 hours of tweet event data (Figure 4.3) and increasing the event length only degraded the performance. Further investigations might be necessary to understand the potential causes of this decreased performance

in the presence of longer events (tweet activity data). It is also important to note that since our goal in this paper is to develop a detection algorithm, investigating the motivation behind the fake news behavior is beyond the scope of this paper. As a future work, we plan to create a man-made disaster fake news dataset as well (e.g., mass shooting, train crashes, chemical pollutant leakage, etc.) to further fine-tune our detection model so that our model could be capable of dealing with diverse types of both natural and man-made disaster fake news situations. In our baseline study, we considered only the textual data of Twitter posts. We plan to expand our study to cover multimedia (such as images and videos) and other embedded URLs in Twitter posts as well. We also plan to experiment with several ideas such as constructing ego networks of Twitter users for the Context Aware module, using different text embedding approaches (e.g., BERT) for tweet texts, etc. to improve our model.

Chapter 5. Conclusion and Summary

This report presents a study on the improvement of the data collection in the data center networks, using Coflourish and MinCOF. It also further analyzes the collected data using deep learning applications, particularly, in studying the fake news propagation during natural disasters.

Coflourish is mainly effective in cloud environment where there are many uncontrolled hosts, leading to a high unknown background traffic. Coflourish achieves 75.5% better average coflow completion time as compared to Varys with various workload conditions. It mainly achieves it by exploiting the congestion feedback from the SDN switches in the networks. Consequently, MinCOF explores the three critical features during the communication between hosts in SANs, which is, coflow scheduling, coflow routing, per-flow per-rate limiting to achieve better coflow completion time. MinCOF decreases the coflow completion time by 12.94% as compared to Tailor. It exploits the OpenFlow SDN to achieve easy deployability, minimum customization and no proprietary framework. Therefore, it demonstrates an effective and faster algorithm which achieves lower coflow completion time both in DCNs or SANs environment.

Finally, the report demonstrates the use of the real time data collected during natural disaster events such as Hurricane Harvey, Hurricane Sandy, etc. to design an end-to-end framework to detect fake news during natural disasters. The model achieves 91% accuracy, when it was trained with the first 28 hours of non-empty partitions. The limitations of small dataset during disaster events are handled using Transfer Learning. Such study provides an automated and easy to use end-to-end data collection framework, as well as, fake events classification model, which can greatly help the emergency managers to continue their rescue and recovery missions during such events without being distrated by misinformation.

Appendix. Copyright Information

1. Coflourish: A Coflow Scheduling Framework for Clouds

The IEEE does not require individuals working on a thesis to obtain a formal reuse license. The IEEE does not endorse any of Louisiana State University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

2. MinCOF: A Coflow Routing and Scheduling Framework for Storage Area Networks

The IEEE does not require individuals working on a thesis to obtain a formal reuse license. The IEEE does not endorse any of Louisiana State University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

3. Identify Fake News During Natural Disasters

17TH INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS FOR CRISIS RESPONSE AND MANAGEMENT

ISCRAM 2020 COPYRIGHT AGREEMENT

This ISCRAM 2020 Proceedings copyright agreement and use license is compliant with the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 ("by-nc-sa") License: see Annex below for terms.

This copyright agreement and use license firstly states that the author(s) keep their copyright and secondly permits any user, for any noncommercial purpose – including unlimited classroom and distance learning use – to download, print out, extract, archive, distribute and make derivative works of an article published in the ISCRAM 2020 Proceedings, as long as appropriate credit is given to the authors and the source of the work and all derivative works are placed under the same license. This copyright agreement and use license ensures, among other things, that an article will be as widely available as possible and that the article can be included in any scientific archive.

COPYRIGHT AGREEMENT

BETWEEN

Dipak Singh

(hereinafter referred to as "the Author", signing as sole author or on behalf of all co-authors)

and

the Proceedings of the ISCRAM 2020 Conference (hereinafter referred to as "the ISCRAM 2020 Proceedings"), represented by Amanda Lee Hughes, Fiona McNeill and Christopher Zobel, the Editors,

regarding the work entitled

Fighting for Information Credibility: An End-to-End Framework to Identify Fake News during Natural Disasters

(Contribution No: 58)
(hereinafter referred to as "the Work"),

submitted to the ISCRAM 2020 Conference and accepted for publication in the ISCRAM 2020 Proceedings.

The Author declares and attests that his or her Work:

1. is distributed under the **Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License** (see Annex below for terms);

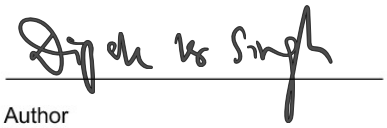
2. is an unpublished and original work and sole property of the Author, who has received any permission required for the purposes of the present agreement from the Work's co-authors and, where applicable, the authors of other works or excerpts presented in the Work;

3. violates no other copyright or proprietary right, and contains no libelous, defamatory or confidential material liable to infringe any law or contractual obligation; and the Author agrees to indemnify and hold harmless the Proceedings from and against any and all legal fees, damages, or other costs resulting from proceedings related to this matter;

The ISCRAM 2020 Proceedings undertake to:

A. publish, in addition to the paper-based Proceedings of ISCRAM2020, the electronic version of the Work online at http://www.iscram.org/ as well as at the ISCRAM Digital Library (http://idl.iscram.org), and enable open access to the Work at all times under the conditions set out in the Creative Commons Attribution 4.0 license cited above;

B. carry out or foster all actions aimed at increasing the Work's visibility and dissemination in all forms and media.



Author

04/20/2020

Date

ANNEX: The Creative Commons Attribution-NonCommercial-ShareAlike 4.0 License with ISCRAM 2020 Proceedings related specifications

You are free:

1. to Share -- to copy, distribute, display, and perform the Work
2. to Remix -- to make derivative works

Under the following conditions:

- Attribution. You must attribute the Work in the manner specified by this copyright agreement and use license (below).
- Noncommercial. You may not use the Work for commercial purposes.
- Share Alike. If you alter, transform, or build upon the Work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of the Work.
- Any of these conditions can be waived if you get permission from the copyright holder - in the case of an ISCRAM 2020 Proceedings paper: its author(s).

The full legal text for this License can be found at https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode

Attribution manner specification: Name Author(s), "Title of the Article", In: Proceedings of the 16th International Conference on Information Systems for Crisis Response and Management ISCRAM 2020 (Eds. Amanda Lee Hughes, Fiona McNeill and Christopher Zobel), 2020, pp. X (startpage) – Y (endpage).

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery.

[2] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 503–514, New York, NY, USA, 2014. Association for Computing Machinery.

[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery.

[4] Amazon.com. Amazon elastic compute cloud. `https://aws.amazon.com/ec2/`. Accessed: 1-2-2017.

[5] Amazon.com. Netflix case study. `https://aws.amazon.com/solutions/case-studies/netflix-case-study/`. 2016.

[6] C.-H. Chiu, N. Lewis, D. K. Singh, A. K. Das, M. M. Jalazai, R. Platania, S. Goswami, K. Lee, and S.-J. Park. Bic-lsu: Big data research integration with cyberinfrastructure for lsu. In *XSEDE16*, 2016.

[7] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, 2014.

[8] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, page 31–36, New York, NY, USA, 2012. Association for Computing Machinery.

[9] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 393–406, New York, NY, USA, 2015. Association for Computing Machinery.

[10] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 98–109, New York, NY, USA, 2011. Association for Computing Machinery.

[11] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 443–454, New York, NY, USA, 2014. Association for Computing Machinery.

[12] R. Community. Ryu sdn framework. `https://ryu.readthedocs.io/en/latest/index.html`. 2017.

[13] R. C. Computing. Openstack. `https://www.openstack.org/`. Accessed: 2-1-2017.

[14] N. J. Conroy, V. L. Rubin, and Y. Chen. Automatic deception detection: Methods for finding fake news. In *Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community*, page 82. American Society for Information Science, 2015.

[15] P. L. Consortium. P4. `https://p4.org/`. Accessed: 12-18-2016.

[16] I. Corp. sflow-rt, 2017. `https://sflow-rt.com/`. Accessed: 1-27-2017.

[17] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[18] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 431–442, New York, NY, USA, 2014. Association for Computing Machinery.

[19] L. Foundation. Open vswitch, 2017. `https://www.openvswitch.org/`. Accessed: 1-15-2017.

[20] O. Foundation. Openstack swift, 2017. `https://docs.openstack.org/swift/latest/`. Accessed: 2-2-2017.

[21] Google. Google cloud platform. `https://cloud.google.com/`. Accessed: 1-15-2017.

[22] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[23] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 51–62, New York, NY, USA, 2009. Association for Computing Machinery.

[24] R. L. Grossman, M. Greenway, A. P. Heath, R. Powell, R. D. Suarez, W. Wells, K. White, M. Atkinson, I. Klampanos, H. L. Alvarez, C. Harvey, and J. J. Mambretti.

[25] A. Hanselowski, A. PVS, B. Schiller, F. Caspelherr, D. Chaudhuri, C. M. Meyer, and I. Gurevych. A retrospective analysis of the fake news challenge stance-detection task. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages

1859–1874, Santa Fe, New Mexico, USA, Aug. 2018. Association for Computational Linguistics.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[27] B. D. Horne and S. Adali. This just in: Fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. *CoRR*, abs/1703.09398, 2017.

[28] S. Hosseinimotlagh and E. E. Papalexakis. Unsupervised content-based identification of fake news articles with tensor decomposition ensembles. *MIS2, Marina Del Rey, CA, USA*, 2018.

[29] J. Jiang, S. Ma, B. Li, and B. Li. Tailor: Trimming coflow completion times in datacenter networks. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2016.

[30] R. E. Kalman. A new approach to linear filtering and prediction problems transaction of the asme journal of basic. 1960.

[31] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[32] Y. Liu and Y. B. Wu. Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 354–361, 2018.

[33] G. K. Lockwood, M. Tatineni, and R. Wagner. Storage utilization in the long tail of science. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, XSEDE '15, New York, NY, USA, 2015. Association for Computing Machinery.

[34] J. Ma, W. Gao, P. Mitra, S. Kwon, B. J. Jansen, K. Wong, and M. Cha. Detecting rumors from microblogs with recurrent neural networks. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3818–3824, 2016.

[35] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 135–146, New York, NY, USA, 2010. Association for Computing Machinery.

[36] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.

[37] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[38] V. Pérez-Rosas, B. Kleinberg, A. Lefevre, and R. Mihalcea. Automatic detection of fake news. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 3391–3401, 2018.

[39] F. Pierri and S. Ceri. False news on social media: A data-driven survey. *CoRR*, abs/1902.07539, 2019.

[40] K. Popat, S. Mukherjee, A. Yates, and G. Weikum. Declare: Debunking fake news and false claims using evidence-aware deep learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 22–32, 2018.

[41] M. Potthast, J. Kiesel, K. Reinartz, J. Bevendorff, and B. Stein. A stylometric inquiry into hyperpartisan and fake news. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 231–240, 2018.

[42] V. Qazvinian, E. Rosengren, D. R. Radev, and Q. Mei. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1589–1599. Association for Computational Linguistics, 2011.

[43] V. L. Rubin, Y. Chen, and N. J. Conroy. Deception detection for news: three types of fakes. In *Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community*, page 83. American Society for Information Science, 2015.

[44] N. Ruchansky, S. Seo, and Y. Liu. Csi: A hybrid deep model for fake news detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 797–806. ACM, 2017.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. 1988.

[46] Selenium. Seleniumhq browser automation. `https://www.seleniumhq.org/`. Accessed: 2019-10-11.

[47] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[48] Snopes.com. Snopes is the internet's definitive fact-checking resource. `https://www.snopes.com/about-snopes/`. Accessed: 2020-10-01.

[49] E. Tacchini, G. Ballarin, M. L. D. Vedova, S. Moret, and L. de Alfaro. Some like it hoax: Automated fake news detection in social networks. *CoRR*, abs/1704.07506, 2017.

[50] Twitter. Twitter ids (snowflakes). `https://developer.twitter.com/en/docs/basics/twitter-ids`. Accessed: 2019-11-16.

[51] S. Volkova and J. Y. Jang. Misleading or falsification: Inferring deceptive strategies and types in online news and social media. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pages 575–583, 2018.

[52] S. Volkova, K. Shaffer, J. Y. Jang, and N. O. Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 647–653, 2017.

[53] S. Vosoughi, D. Roy, and S. Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[54] W. Y. Wang. "Liar, Liar Pants on Fire": A new benchmark dataset for fake news detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 422–426, 2017.

[55] Y. Wang, F. Ma, Z. Jin, Y. Yuan, G. Xun, K. Jha, L. Su, and J. Gao. EANN: Event adversarial neural networks for multi-modal fake news detection. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 849–857, 2018.

[56] L. Wu and H. Liu. Tracing fake-news footprints: Characterizing social media messages by how they propagate. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 637–645, 2018.

[57] L. Xue, C. Chiu, S. Kumar, P. Kondikoppa, and S. Park. Fall: A fair and low latency queuing scheme for data center networks. In *2015 International Conference on Computing, Networking and Communications (ICNC)*, pages 771–777, 2015.

[58] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 424–432, 2015.

Vita

Dipak Kumar Singh, born in West Bengal, India, received his Bachelor's degree in Computer Science and Engineering from the National Institute of Technology, Durgapur. As a PhD student, he worked as a graduate assistant at the Department of Computer Science and Engineering, Louisiana State University, Baton Rouge. His research interests are big data, data center networks and machine learning, specifically in the field of natural disaster resilience and also in bridging the gap between data center networks and deep learning.